软件开发微视频讲堂

核心技术分册

微视频精编版

明日科技 编著

SQL Server 从入门到精通

71集高清微课视频，随时可学
217个应用实例
6个企业项目案例

在玩中嗨翻SQL Server
趣味解读 / 易学易用 / 学练结合
知识点讲解+实例+项目

海量资源，可查可练
- 实例资源库
- 模块资源库
- 项目资源库
- 测试题库系统

清華大學出版社

软件开发微视频讲堂

SQL Server 从入门到精通

（微视频精编版）

明日科技　编著

清華大学出版社

北　京

内 容 简 介

本书内容浅显易懂，实例丰富，详细介绍了从基础入门到SQL Server数据库高手需要掌握的知识。

全书分为上下两册：核心技术分册和项目实战分册。核心技术分册共2篇19章，包括数据库基础、SQL Server 2014安装与配置、创建和管理数据库、操作数据表、操作表数据、SQL函数的使用、视图操作、Transact-SQL语法基础、数据的查询、子查询与嵌套查询、索引与数据完整性、流程控制、存储过程、触发器、游标的使用、SQL中的事务、SQL Server高级开发、SQL Server安全管理和SQL Server维护管理等内容。项目实战分册共6章，运用软件工程的设计思想，介绍了腾宇超市管理系统、学生成绩管理系统、图书商城、房屋中介管理系统、客房管理系统和在线考试系统共6个完整企业项目的真实开发流程。

本书除纸质内容外，配书资源包中还给出了海量开发资源，主要内容如下。

- ☑ **微课视频讲解：总时长8小时，共71集**
- ☑ **实例资源库：126个实例及源码分析**
- ☑ **模块资源库：15个经典模块完整展现**
- ☑ **项目案例资源库：15个企业项目开发过程**
- ☑ **测试题库系统：596道能力测试题目**

本书适合有志于从事软件开发的初学者、高校计算机相关专业学生和毕业生，也可作为软件开发人员的参考手册，或者高校的教学参考书。

图书在版编目（CIP）数据

SQL Server从入门到精通 : 微视频精编版/明日科技编著. —北京：清华大学出版社，2020.7
（软件开发微视频讲堂）
ISBN 978-7-302-52090-0

Ⅰ. ①S… Ⅱ. ①明… Ⅲ. ①关系数据库系统 Ⅳ. ①TP311.132.3

中国版本图书馆CIP数据核字（2019）第010403号

责任编辑：贾小红
封面设计：魏润滋
版式设计：文森时代
责任校对：马军令
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：
经　　销：全国新华书店
开　　本：203mm×260mm　　**印　　张**：38　　**字　　数**：1025千字
版　　次：2020年8月第1版　　**印　　次**：2020年8月第1次印刷
定　　价：99.80元（全两册）

产品编号：079180-01

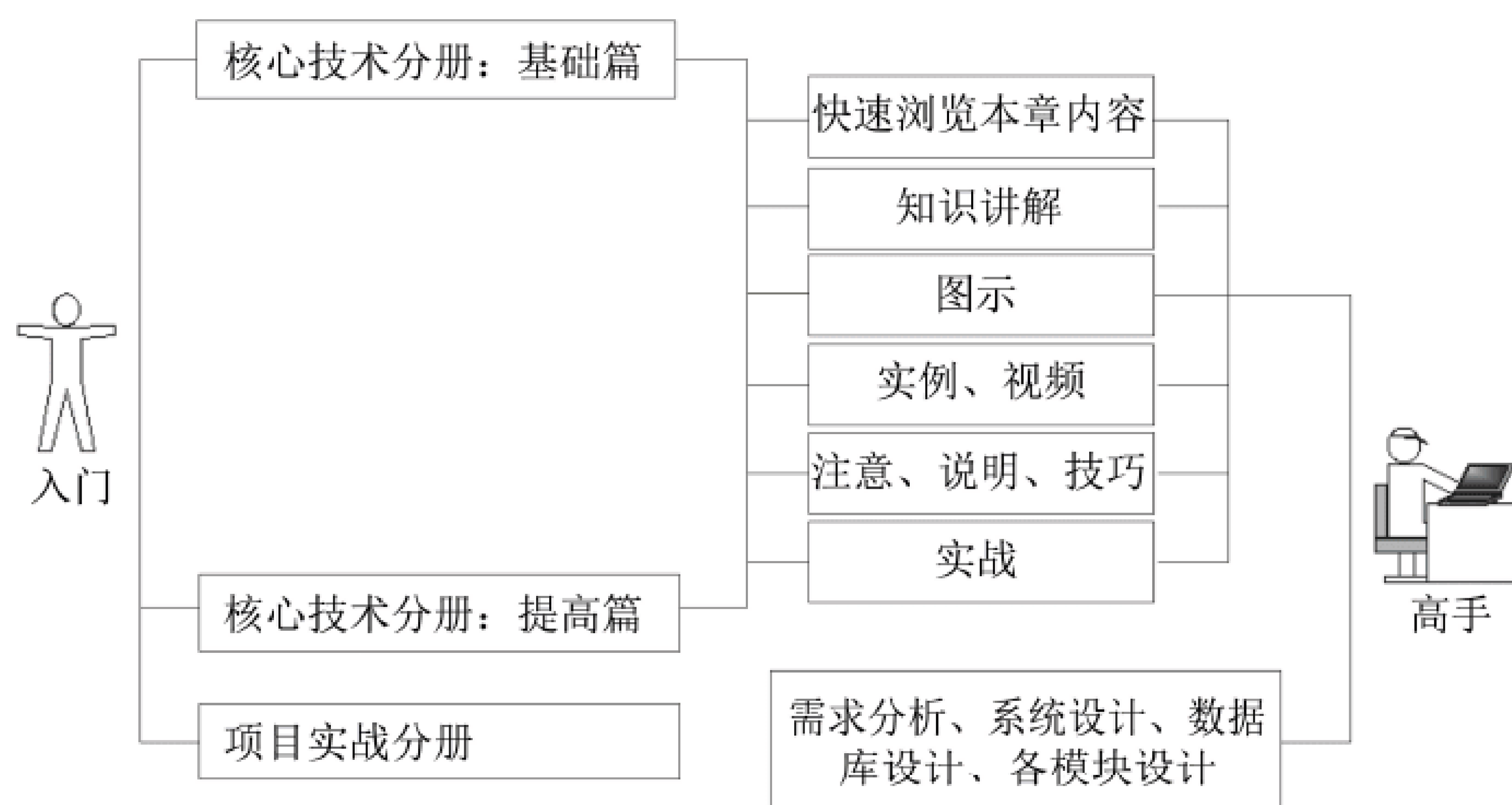
前　言

Preface

SQL Server 是由美国微软（Microsoft）公司制作并发布的一种性能优越的关系型数据库管理系统（Relational Database Management System，RDBMS），因其具有良好的数据库设计、管理与网络功能，又与 Windows 系统紧密集成，因此成为数据库产品的首选。

本书内容

本书分上下两册，上册为核心技术分册，下册为项目实战分册，大体结构如下图所示。



核心技术分册共分 2 篇 19 章，提供了从基础入门到 SQL Server 数据库高手所必备的各类知识。

基础篇：介绍了通过数据库基础、SQL Server 2014 安装与配置、创建和管理数据库、操作数据表、操作表数据、SQL 函数的使用、视图操作、Transact-SQL 语法基础、数据的查询、子查询与嵌套查询等内容，并结合大量的图示、实例、视频和实战等，使读者快速掌握 SQL 语言基础。

提高篇：介绍了索引与数据完整性、流程控制、存储过程、触发器、游标的使用、SQL 中的事务、SQL Server 高级开发、SQL Server 安全管理和 SQL Server 维护管理等内容。学习完本篇，能够掌握比较高级的 SQL 及 SQL Server 管理知识，并对数据库进行管理。

项目实战分册共 6 章，运用软件工程的设计思想，介绍了 6 个完整企业项目（腾宇超市管理系统、学生成绩管理系统、图书商城、房屋中介管理系统、客房管理系统和在线考试系统）的真实开发流程。书中按照“需求分析→系统设计→数据库设计→项目主要功能模块的实现”的流程进行介绍，带领读者亲身体验开发项目的全过程，提升实战能力，实现从小白到高手的跨越。

本书特点

☑ **由浅入深，循序渐进**。本书以初、中级读者为对象，先从 SQL 语言基础学起，再学习数据库对象的使用，如视图、存储过程、触发器等，最后学习开发一个完整项目。讲解过程中步骤详尽，版式新颖，使读者在阅读时一目了然，从而快速掌握书中内容。

☑ **实例典型，轻松易学**。通过例子学习是最好的学习方式，本书通过“一个知识点、一个例子、一个结果、一段评析，一个综合应用”的模式，透彻详尽地讲述了实际开发中所需的各类知识。另外，为了便于读者阅读程序代码，快速学习编程技能，书中绝大多数代码提供了注释。

☑ **微课视频，讲解详尽**。本书为便于读者直观感受程序开发的全过程，书中大部分章节都配备了教学微视频，使用手机扫描正文小节标题一侧的二维码，即可观看学习，能快速引导初学者入门，感受编程的快乐和成就感，进一步增强学习的信心。

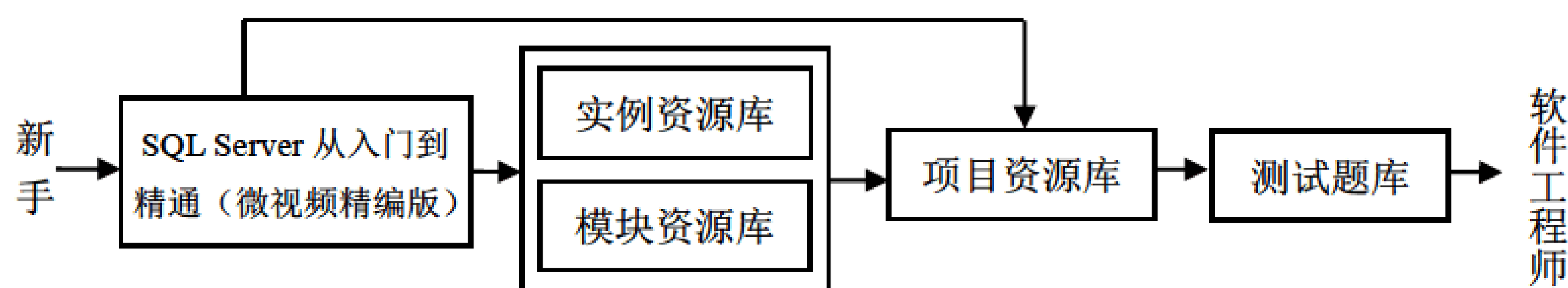
☑ **精彩栏目，贴心提醒**。本书根据需要在各章安排了“注意”“说明”等小栏目，让读者可以在学习过程中更轻松地理解相关知识点及概念，更快地掌握个别技术的应用技巧。

☑ **紧跟潮流，流行技术**。本书采用使用广泛的数据库版本——SQL Server 2014 实现，使读者能够紧跟技术发展的脚步。

本书资源

为帮助读者学习，本书配备了长达 8 个小时（共 71 集）的微课视频讲解。除此以外，还为读者提供了“ASP.NET+SQL Server 自主学习系统”，可以帮助读者快速提升编程水平和解决实际问题的能力。

本书和“ASP.NET + SQL Server 自主学习系统”配合学习流程如图所示。



“ASP.NET + SQL Server 自主学习系统”的主界面如下图所示。





在学习本书的过程中，可以选择实例资源库和项目资源库的相应内容，全面提升个人综合编程技能和解决实际开发问题的能力，为成为软件开发工程师打下坚实基础。

对于数学及逻辑思维能力和英语基础较为薄弱的读者，或者想了解个人数学及逻辑思维能力和编程英语基础的用户，本书提供了数学及逻辑思维能力测试和编程英语能力测试供练习和测试。

读者对象

- ☑ 初学编程的自学者
- ☑ 编程爱好者
- ☑ 大中专院校的老师和学生
- ☑ 相关培训机构的老师和学员
- ☑ 做毕业设计的学生
- ☑ 初、中级程序开发人员
- ☑ 程序测试及维护人员
- ☑ 参加实习的“菜鸟”程序员

读者服务

学习本书时，请先扫描封底的权限二维码（需要刮开涂层）获取学习权限，然后即可免费学习书中的所有线上线下资源。本书所附赠的各类学习资源，读者可登录清华大学出版社网站（www.tup.com.cn），在对应图书页面下获取其下载方式。也可扫描图书封底的“文泉云盘”二维码，获取其下载方式。

致读者

本书由明日科技程序开发团队组织编写，明日科技是一家专业从事软件开发、教育培训以及软件开发教育资源整合的高科技公司，其编写的教材既注重选取软件开发中的必需、常用内容，又注重内容的易学、方便以及相关知识的拓展，深受读者喜爱。其编写的教材多次荣获“全行业优秀畅销品种”“中国大学出版社优秀畅销书”等奖项，多个品种长期位居同类图书销售排行榜的前列。在编写过程中，我们以科学、严谨的态度，力求精益求精，但错误、疏漏之处在所难免，敬请广大读者批评指正。

感谢您购买本书，希望本书能成为您编程路上的领航者。

“零门槛”编程，一切皆有可能。

祝读书快乐！

编　者

2020 年 7 月

目 录

Contents

第 1 篇 基 础 篇

第 2 篇 提 高 篇

第1篇

基础篇

本篇通过数据库基础、SQL Server 2014 安装与配置、创建和管理数据库、操作数据表、操作表数据、SQL 函数的使用、视图操作、Transact-SQL 语法基础、数据的查询、子查询与嵌套查询等内容的介绍，并结合大量的图示、实例和视频等，使读者快速掌握 SQL 语言基础。

第 1 章

数据库基础

（视频讲解：28 分钟）

本章主要介绍数据库的相关概念，包括数据库系统简介、数据库的体系结构、数据模型、常见关系数据库及Transact-SQL简介。通过本章的学习，读者应该掌握数据库系统、数据库三级模式结构、数据模型及数据库规范化等概念，对比常见的关系数据库，了解Transact-SQL语言。

学习摘要：

- **数据库系统简介**
- **数据库的体系结构**
- **常见的数据模型**
- **常见的关系数据库**

1.1　数据库系统简介



1.1.1　数据库技术的发展

数据库技术是应数据管理任务的需求而产生的。随着计算机技术的发展，对数据管理技术也不断地提出更高的要求，其先后经历了人工管理、文件系统、数据库系统 3 个阶段，下面分别对这 3 个阶段进行介绍。

1. 人工管理阶段

20 世纪 50 年代中期以前，计算机主要用于科学计算。当时硬件和软件设备都很落后，数据基本依赖于人工管理。人工管理数据具有如下特点。

（1）数据不保存。

（2）使用应用程序管理数据。

（3）数据不共享。

（4）数据不具有独立性。

2. 文件系统阶段

20 世纪 50 年代后期到 60 年代中期，硬件和软件技术都有了进一步发展，有了磁盘等存储设备和专门的数据管理软件即文件系统，其具有如下特点。

（1）数据可以长期保存。

（2）由文件系统管理数据。

（3）共享性差，数据冗余大。

（4）数据独立性差。

3. 数据库系统阶段

20 世纪 60 年代后期以来，计算机应用于管理系统，而且规模越来越大，应用越来越广泛，数据量急剧增长，对共享功能的要求越来越强烈。这样使用文件系统管理数据已经不能满足要求，于是为了解决一系列问题，出现了数据库系统，用来统一管理数据。其满足了多用户、多应用共享数据的需求，比文件系统具有更明显的优点，标志着管理技术的飞跃。

1.1.2　数据库系统的组成

数据库系统（Database System，DBS）是采用数据库技术的计算机系统，是由数据库（数据）、数据库管理系统（软件）、数据库管理员（人员）、硬件平台（硬件）和软件平台（软件）5 部分构成的运行实体。其中，数据库管理员（Database Administrator，DBA）是对数据库进行规划、设计、维护和监视等操作的专业管理人员，在数据库系统中起着非常重要的作用。



1.2　数据库的体系结构

数据库具有一个严谨的体系结构，这样可以有效地组织、管理数据，提高数据库的逻辑独立性和物理独立性。数据库领域公认的标准结构是三级模式结构。

1.2.1　数据库三级模式结构

数据库系统的三级模式结构是指模式、外模式和内模式。下面分别进行介绍。

1．模式

模式也称逻辑模式或概念模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。一个数据库只有一个模式。模式处于三级结构的中间层。


注意

定义模式时不仅要定义数据的逻辑结构，而且要定义数据之间的联系，定义与数据有关的安全性、完整性要求。

2．外模式

外模式也称用户模式，它是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。外模式是模式的子集，一个数据库可以有多个外模式。


说明

外模式是保证数据安全性的一个有力措施。

3．内模式

内模式也称存储模式，一个数据库只有一个内模式。它是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式。

1.2.2　三级模式之间的映射

为了能够在内部实现数据库的 3 个抽象层次的联系和转换，数据库管理系统在三级模式之间提供了两层映射，分别为外模式/模式映射和模式/内模式映射，下面分别介绍。

1．外模式/模式映射

同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式/模式映射。当模式改变时，由数据库管理员对各个外模式/模式映射做相应的改变，可以使外模式保持不变。这样，

依据数据外模式编写的应用程序就不用修改，其保证了数据与程序的逻辑独立性。

2. 模式/内模式映射

数据库中只有一个模式和一个内模式，所以模式/内模式映射是唯一的，它定义了数据库的全局逻辑结构与存储结构之间的对应关系。当数据库的存储结构改变时，由数据库管理员对模式/内模式映射进行相应的改变，可以使模式保持不变，应用程序也相应地不变动。这样，保证了数据与程序的物理独立性。

1.3　数 据 模 型

视频讲解

1.3.1　数据模型的概念

数据模型是数据库系统的核心与基础，是描述数据与数据之间的联系、数据的语义、数据一致性约束的概念性工具的集合。

数据模型通常是由数据结构、数据操作和完整性约束 3 部分组成的，分别如下。

（1）数据结构：是对系统静态特征的描述，描述对象包括数据的类型、内容、性质和数据之间的相互关系。

（2）数据操作：是对系统动态特征的描述，是对数据库中各种对象实例的操作。

（3）完整性约束：是完整性规则的集合。它定义了给定数据模型中数据及其联系所具有的制约和依存规则。

1.3.2　常见的数据模型

常用的数据库数据模型主要有层次模型、网状模型和关系模型，下面分别进行介绍。

（1）层次模型：用树型结构表示实体类型及实体间联系的数据模型称为层次模型，如图 1.1 所示，它具有以下特点。

① 每棵树有且仅有一个无双亲节点，称为根。

② 树中除根外的所有节点有且仅有一个双亲。

（2）网状模型：用有向图结构表示实体类型及实体间联系的数据模型称为网状模型，如图 1.2 所示。用网状模型编写的应用程序极其复杂，且数据的独立性较差。

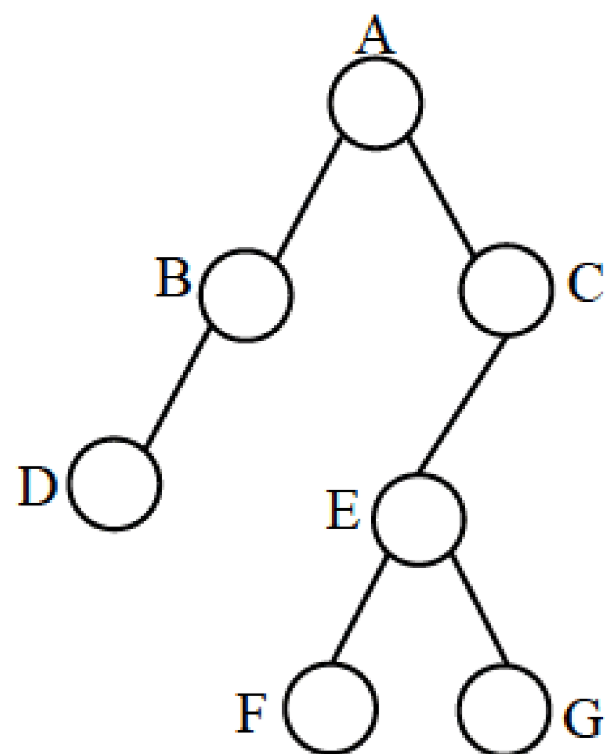


图 1.1　层次模型

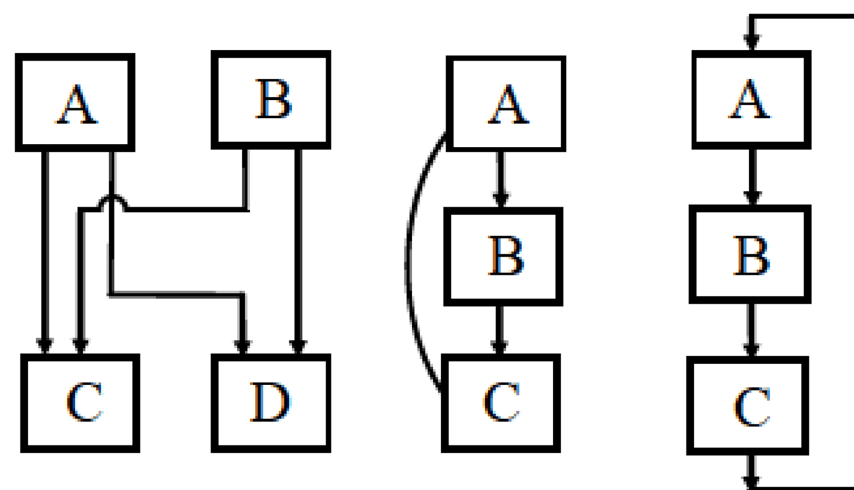


图 1.2　网状模型

（3）关系模型：以二维表来描述数据，如图 1.3 所示。在关系模型中，每个表有多个字段列和记录行，每个字段列有固定的属性（数字、字符、日期等）。关系模型的数据结构简单、清晰、具有很高的数据独立性，因此是目前主流的数据库数据模型。

学生信息表

学生姓名	年级	家庭住址
张三	2000	成都
李四	2000	北京
王五	2000	上海

成绩表

学生姓名	课程	成绩
张三	数学	100
张三	物理	95
张三	社会	90
李四	数学	85
李四	社会	90
王五	数学	80
王五	物理	75

图 1.3　关系模型

关系模型的基本术语如下。

① 关系：一个二维表就是一个关系。

② 元组：就是二维表中的一行，即表中的记录。

③ 属性：就是二维表中的一列，用类型和值表示。

④ 域：每个属性取值的变化范围，如性别的域为{男，女}。

关系中的数据约束如下。

① 实体完整性约束：约束关系的主键中属性值不能为空值。

② 参照完整性约束：关系之间的基本约束。

③ 用户定义的完整性约束：它反映了具体应用中数据的语义要求。

1.3.3　关系数据库的规范化

关系数据库的规范化理论认为：关系数据库中的每一个关系都要满足一定的规范。根据满足规范的条件不同，可以分为 5 个等级：第一范式（1NF）、第二范式（2NF）……第五范式（5NF）。其中，NF 是 Normal Form 的缩写。一般情况下，只要把数据规范到第三个范式标准就可以满足需要了。

（1）第一范式（1NF）：在一个关系中，消除重复字段，且各字段都是最小的逻辑存储单位。

（2）第二范式（2NF）：若关系模型属于第一范式，则关系中每一个非主关键字段都完全依赖于主关键字段，不能只部分依赖于主关键字的一部分。

（3）第三范式（3NF）：若关系属于第一范式，且关系中所有非主关键字段都只依赖于主关键字

段，第三范式要求去除传递依赖。

1.3.4　关系数据库的设计原则

数据库设计是指对于一个给定的应用环境，根据用户的需求，利用数据模型和应用程序模拟现实世界中该应用环境的数据结构和处理活动的过程。

数据库设计原则如下。

（1）数据库内数据文件的数据组织应获得最大限度的共享、最小的冗余度，消除数据及数据依赖关系中的冗余部分，使依赖于同一个数据模型的数据达到有效的分离。

（2）保证输入、修改数据时数据的一致性与正确性。

（3）保证数据与使用数据的应用程序之间的高度独立性。

1.3.5　实体与关系

实体是指客观存在并可相互区别的事物，实体既可以是实际的事物，也可以是抽象的概念或关系。

实体之间有 3 种关系，分别如下。

（1）一对一关系：是指表 A 中的一条记录确实在表 B 中有且只有一条相匹配的记录。在一对一关系中，大部分相关信息都在一个表中。

（2）一对多关系：是指表 A 中的行可以在表 B 中有许多匹配行，但是表 B 中的行只能在表 A 中有一个匹配行。

（3）多对多关系：是指关系中每个表的行在相关表中具有多个匹配行。在数据库中，多对多关系的建立是依靠第 3 个表（称作连接表）实现的，连接表包含相关的两个表的主键列，然后从两个相关表的主键列分别创建与连接表中的匹配列的关系。


视频讲解

1.4　常见关系数据库

1.4.1　Access 数据库

Access 是当前流行的关系型数据库管理系统之一，其核心是 Microsoft Jet 数据库引擎。通常情况下，安装 Microsoft Office 时选择默认安装，Access 数据库即被安装到计算机上。

Access 是一个非常容易掌握的数据库管理系统。利用它可以创建、修改和维护数据库及数据库中的数据，并且可以利用向导来完成对数据库的一系列操作。Access 能够满足小型企业客户/服务器解决方案的要求，是一种功能较完备的系统，它几乎包含了数据库领域的所有技术和内容，对于初学者学习数据库知识非常有帮助。

1.4.2　SQL Server 数据库

SQL Server 是由微软（Microsoft）公司开发的一个大型的关系数据库系统，它为用户提供了一个

安全、可靠、易管理和高端的客户/服务器数据库平台。

SQL Server 数据库有很多版本，如 SQL Server 2000、SQL Server 2008、SQL Server 2012、SQL Server 2014 等。各版本发布时间如图 1.4 所示。

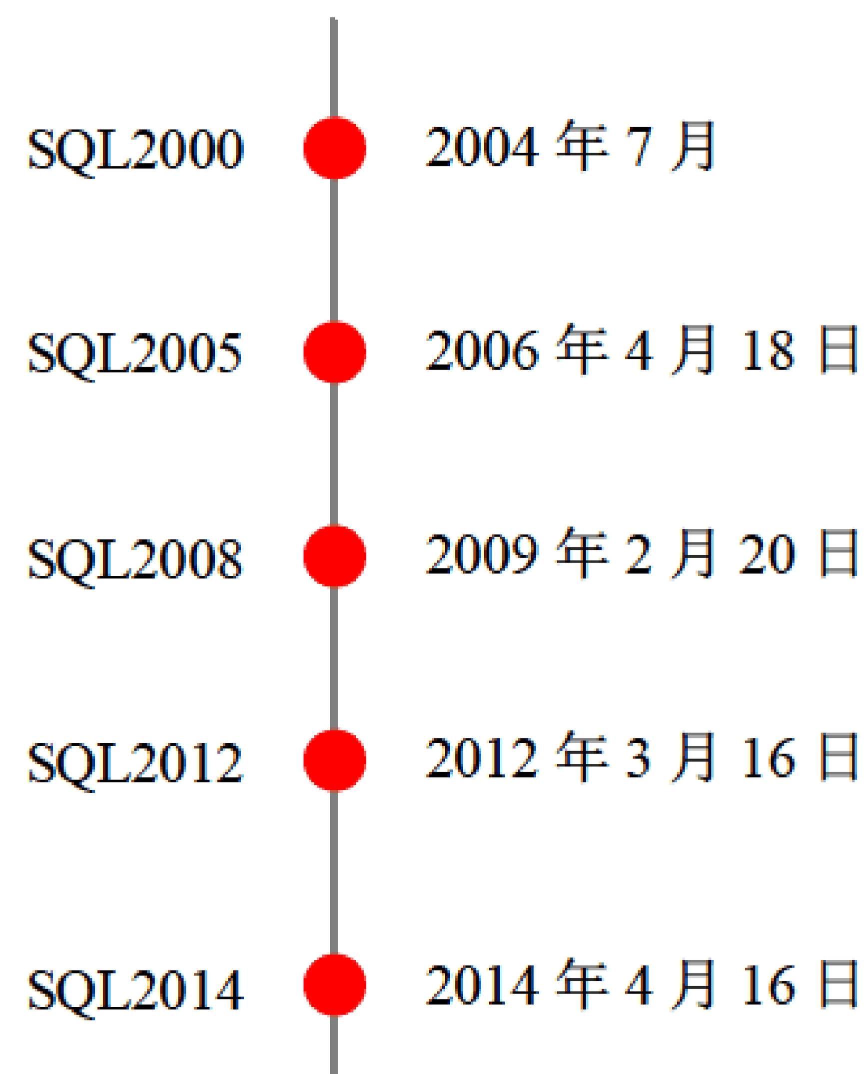


图 1.4　各版本发布时间

1.4.3　Oracle 数据库

Oracle 是甲骨文（ORACLE）公司提供的以分布式数据库为核心的一组软件产品。Oracle 是目前世界上使用最为广泛的关系型数据库。它具有完整的数据管理功能，包括数据的大量性、数据保存的持久性、数据的共享性、数据的可靠性。

Oracle 在并行处理、实时性、数据处理速度方面都有较好的表现。一般情况下，大型企业选择 Oracle 作为后台数据库来处理海量数据。



1.5　Transact-SQL 简介

Transact-SQL 是 SQL Server 2008 在 SQL 基础上添加了流程控制语句后的扩展，是标准的 SQL 的超集，简称 T-SQL。

SQL 是关系数据库系统的标准语言，标准的 SQL 语句几乎可以在所有的关系型数据库上不加修改地使用。Access、Oracle 这样的数据库同样支持标准的 SQL，但这些关系数据库不支持 Transact-SQL。Transact-SQL 是 SQL Server 系统产品独有的。

1．Transact-SQL 语法

Transact-SQL 的语法规则如表 1.1 所示。

表 1.1　T-SQL 语法规则

约　　定	说　　明
UPPERCASE（大写）	T-SQL 关键字
Italic（斜体）	用户提供的 T-SQL 语法的参数
Bold（粗体）	数据库名、表名、列名、索引名、存储过程、实用工具、数据类型名以及必须按所显示的原样键入的文本
下画线	指示当语句中省略了包含带下画线的值的子句时应用的默认值
|（竖线）	分隔括号或大括号中的语法项。只能选择其中一项
[ ]（方括号）	可选语法项。不要输入方括号
{ }（大括号）	必选语法项。不要输入大括号
[,...n]	指示前面的项可以重复 n 次。每一项由逗号分隔
[...n]	指示前面的项可以重复 n 次。每一项由空格分隔
[;]	可选的 T-SQL 语句终止符。不要输入方括号
<label> :: =	语法块的名称。此约定用于对可在语句中的多个位置使用的过长语法段或语法单元进行分组和标记。可使用的语法块的每个位置应括在尖括号内

2. Transact-SQL 语言分类

Transact-SQL 语言的分类如下。

（1）变量说明语句：用来说明变量的命令。

（2）数据定义语言：用来建立数据库、数据库对象和定义列，大部分是以 CREATE 开头的命令，如 CREATE TABLE、CREATE VIEW 和 DROP TABLE 等。

（3）数据操纵语言：用来操纵数据库中数据的命令，如 SELECT、INSERT、UPDATE、DELETE 和 CURSOR 等。

（4）数据控制语言：用来控制数据库组件的存取许可、存取权限等命令。

（5）流程控制语言：用于设计应用程序流程的语句，如 IF WHILE 和 CASE 等。

（6）内嵌函数：实现参数化视图的功能。

（7）其他命令：嵌于命令中使用的标准函数。

1.6　小　　结

本章介绍了数据库的基本概念：数据库系统的组成、数据库三级模式结构及映射、关系数据库的规范化及设计原则等。通过本章的学习，读者可以对数据库有一个系统的了解，在此基础上了解 Transact-SQL 语言，为进一步的学习奠定基础。

第 2 章

SQL Server 2014 安装与配置

（视频讲解：11 分钟）

本章内容包括 SQL Server 2014 简介、安装 SQL Server 2014、启动 SQL Server 2014 管理工具、脚本与批处理，以及数据库的备份和还原、分离和附加、导入和导出。通过本章的学习，读者应该熟悉 SQL Server 2014，选择合适的版本进行安装和配置，并掌握操作 SQL Server 2014 数据库的方法等。

学习摘要：

- SQL Server 简介
- 安装 SQL Server 2014
- 脚本与批处理
- 备份和还原数据库
- 分离和附加数据库
- 导入和导出数据库或数据表

2.1　SQL Server 数据库简介

SQL Server 是由微软（Microsoft）公司开发的一个大型的关系数据库系统，它为用户提供了一个安全、可靠、易管理和高端的客户/服务器数据库平台。

SQL Server 数据库的中心数据驻留在一个中心计算机上，该计算机被称为服务器。用户通过客户机的应用程序来访问服务器上的数据库，在被允许访问数据库之前，SQL Server 首先对来访问的用户请求做安全验证，只有验证通过后才能够进行处理请求，并将处理的结果返回给客户机应用程序。


视频讲解

2.2　安装 SQL Server

SQL Server 是微软公司推出的数据库服务器工具，从最初的 SQL Server 2000 版本起步，逐渐发展到至今的 SQL Server 2017，深受广大开发者的喜爱。从 SQL Server 2005 版本之后，SQL Server 数据库的安装与配置过程类似，这里以 SQL Server 2014 版本为例讲解 SQL Server 数据库的安装与配置过程。

2.2.1　SQL Server 2014 安装必备

安装 SQL Server 2014 之前，首先要了解安装所需的必备条件，检查计算机的软硬件配置是否满足 SQL Server 2014 的安装要求，具体要求如表 2.1 所示。

表 2.1　安装 SQL Server 2014 所需的必备条件

名　称	说　明
操作系统	Windows 7（SP1）、Windows 8、Windows 8.1、Windows Server 2008 R2 SP1（x64）、Windows Server 2012（x64）、Windows 10
软件	SQL Server 安装程序需要使用 Microsoft Windows Installer 4.5 或更高版本以及 Microsoft 数据访问组件（MDAC）2.8 SP1 或更高版本
处理器	1.4GHz 处理器，建议使用 2.0GHz 或速度更快的处理器
内存	最小 2GB，建议使用 4GB 或更大的内存
可用硬盘空间	至少 2.2GB 的可用硬盘空间
驱动器	从磁盘进行安装时需要相应的 DVD 驱动器
显示器	SQL Server 2014 要求有 Super-VGA（800×600）或更高分辨率的显示器

2.2.2　SQL Server 2014 的安装

安装 SQL Server 2014 数据库的步骤如下。

（1）使用虚拟光驱软件加载下载的 SQL Server 2014 的安装镜像文件（.iso 文件），在“SQL Server

安装中心”窗口中单击左侧的“安装”选项，再单击“全新 SQL Server 独立安装或向现有安装添加功能”超链接，如图 2.1 所示。

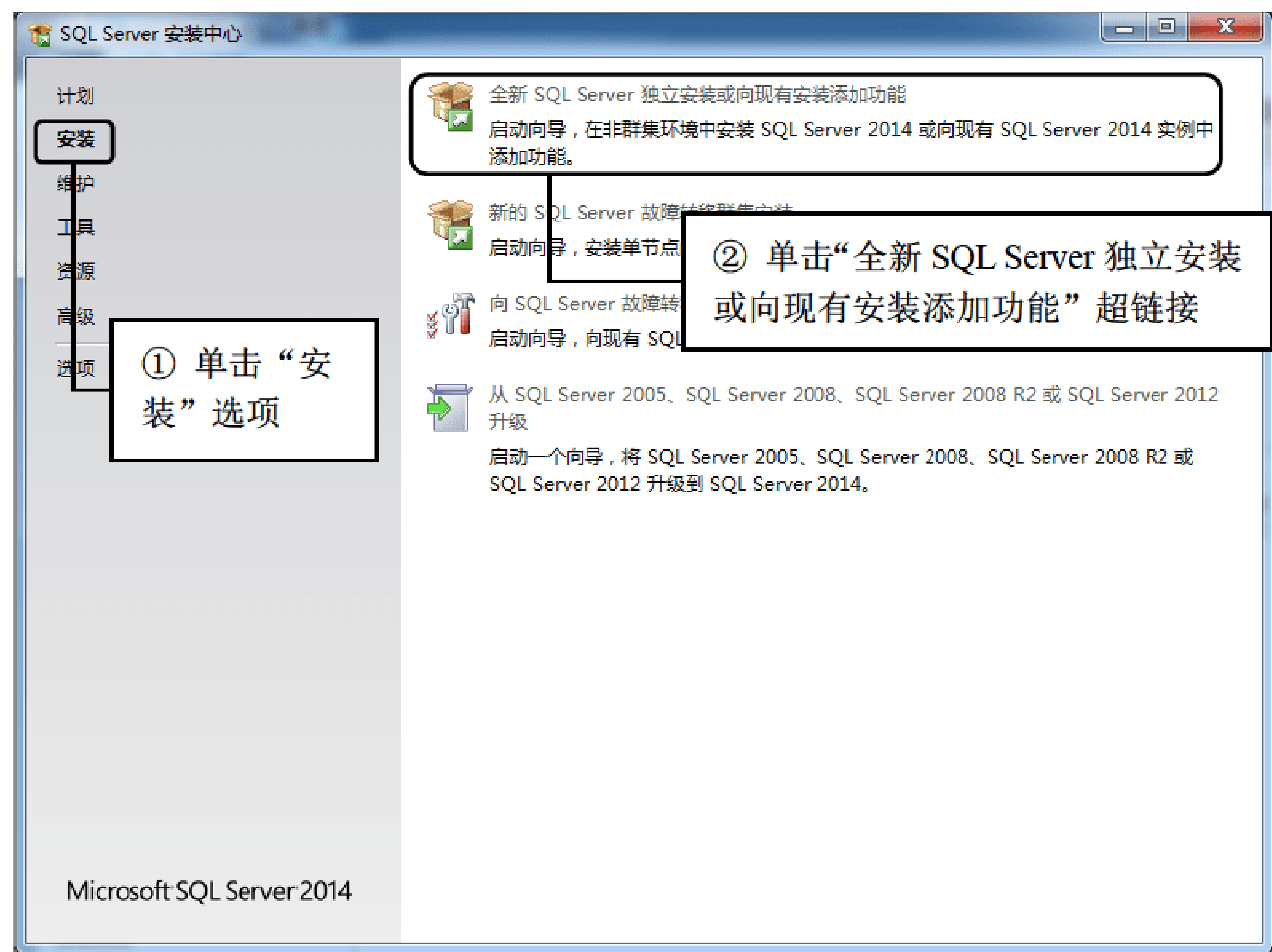


图 2.1　单击左侧的“安装”选项

（2）单击“下一步”按钮，打开“产品密钥”界面，如图 2.2 所示，该界面中输入产品密钥。

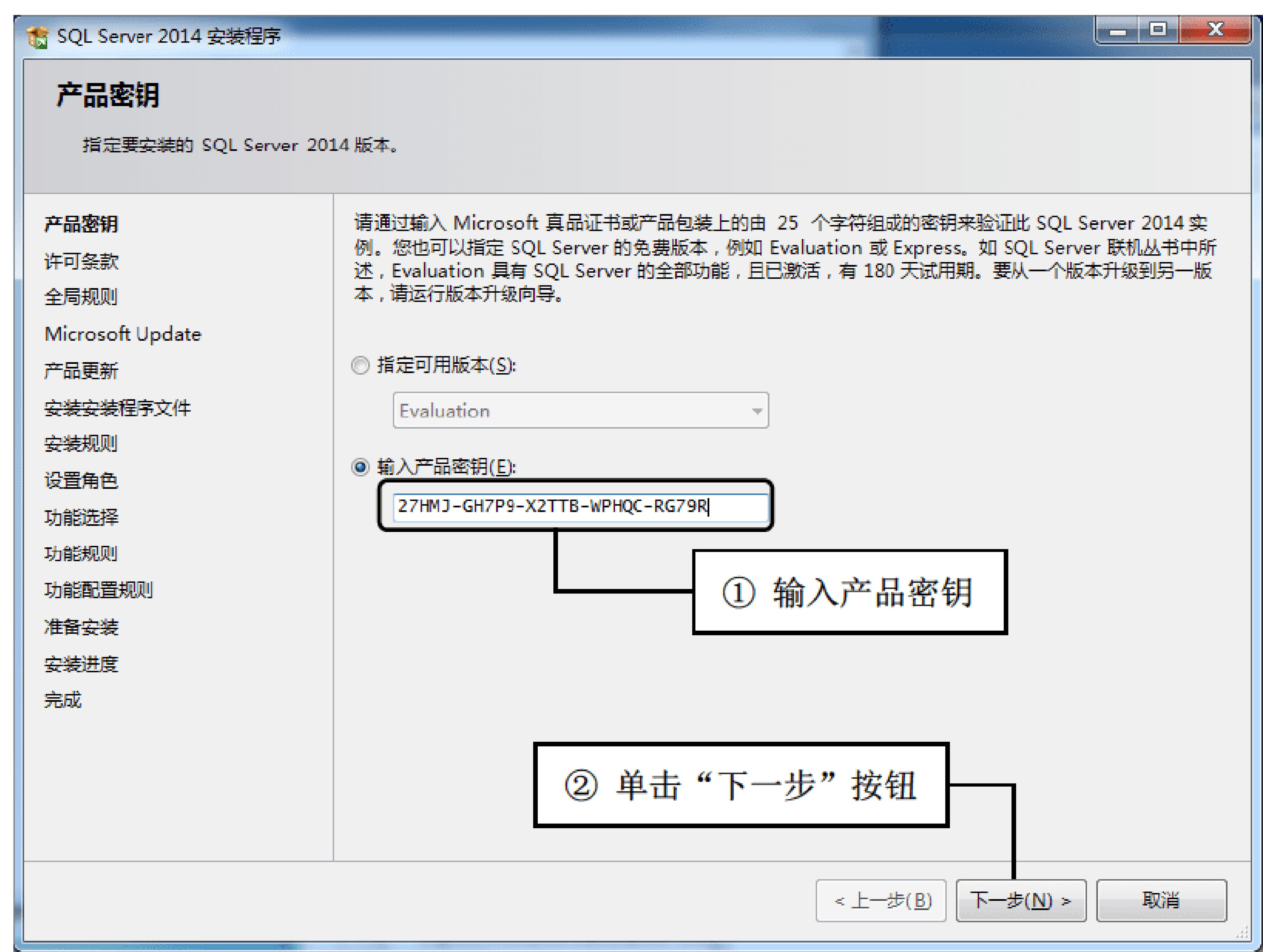


图 2.2　“产品密钥”界面

（3）单击“下一步”按钮，进入“许可条款”界面，如图 2.3 所示，选中“我接受许可条款”复选框。

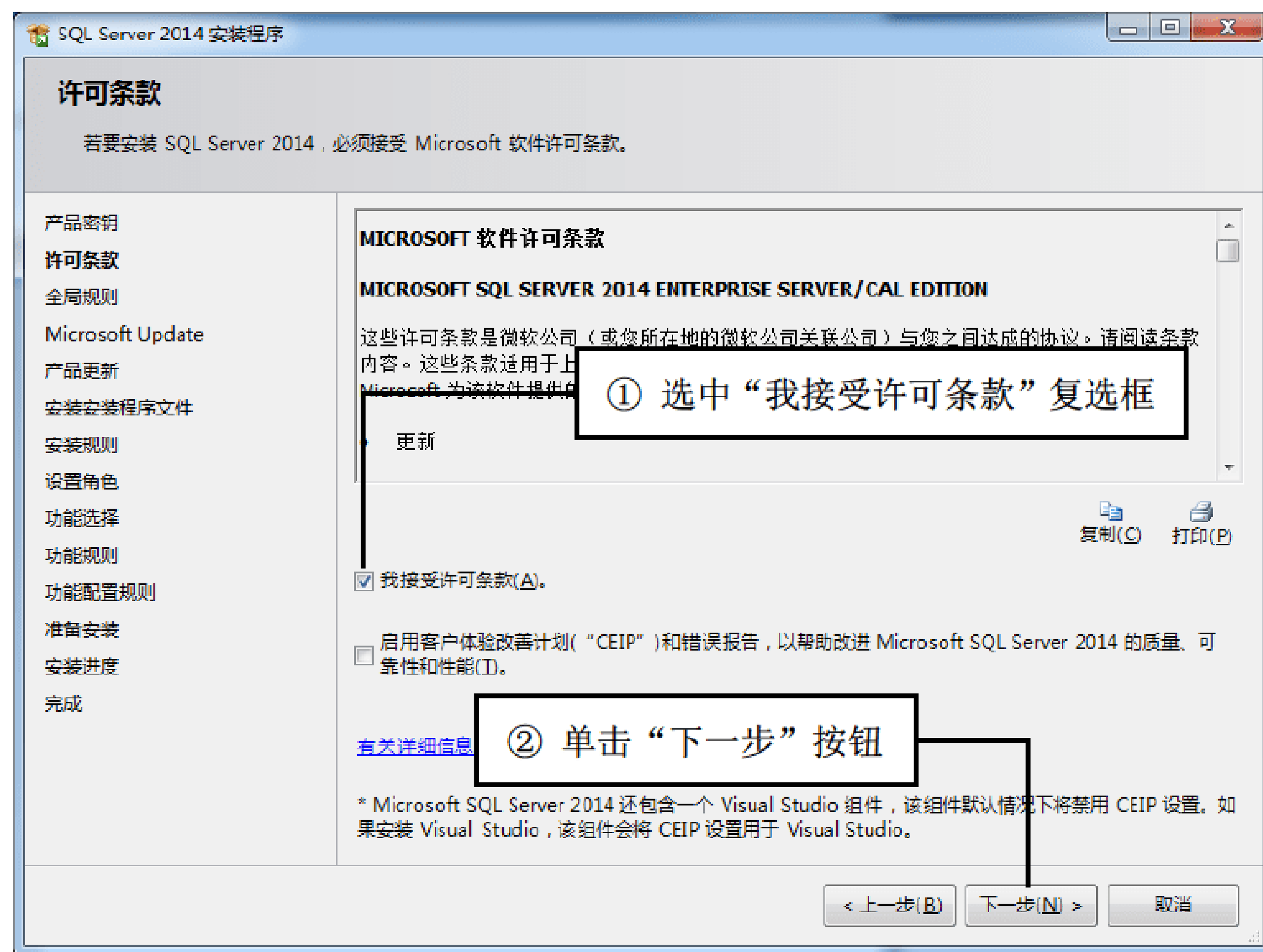


图 2.3　“许可条款”界面

（4）单击“下一步”按钮，进入“全局规则”界面，规则检查完成后，“下一步”按钮可用，如图 2.4 所示。



图 2.4　“全局规则”界面

（5）单击“下一步”按钮，进入 Microsoft Update 界面，如图 2.5 所示。

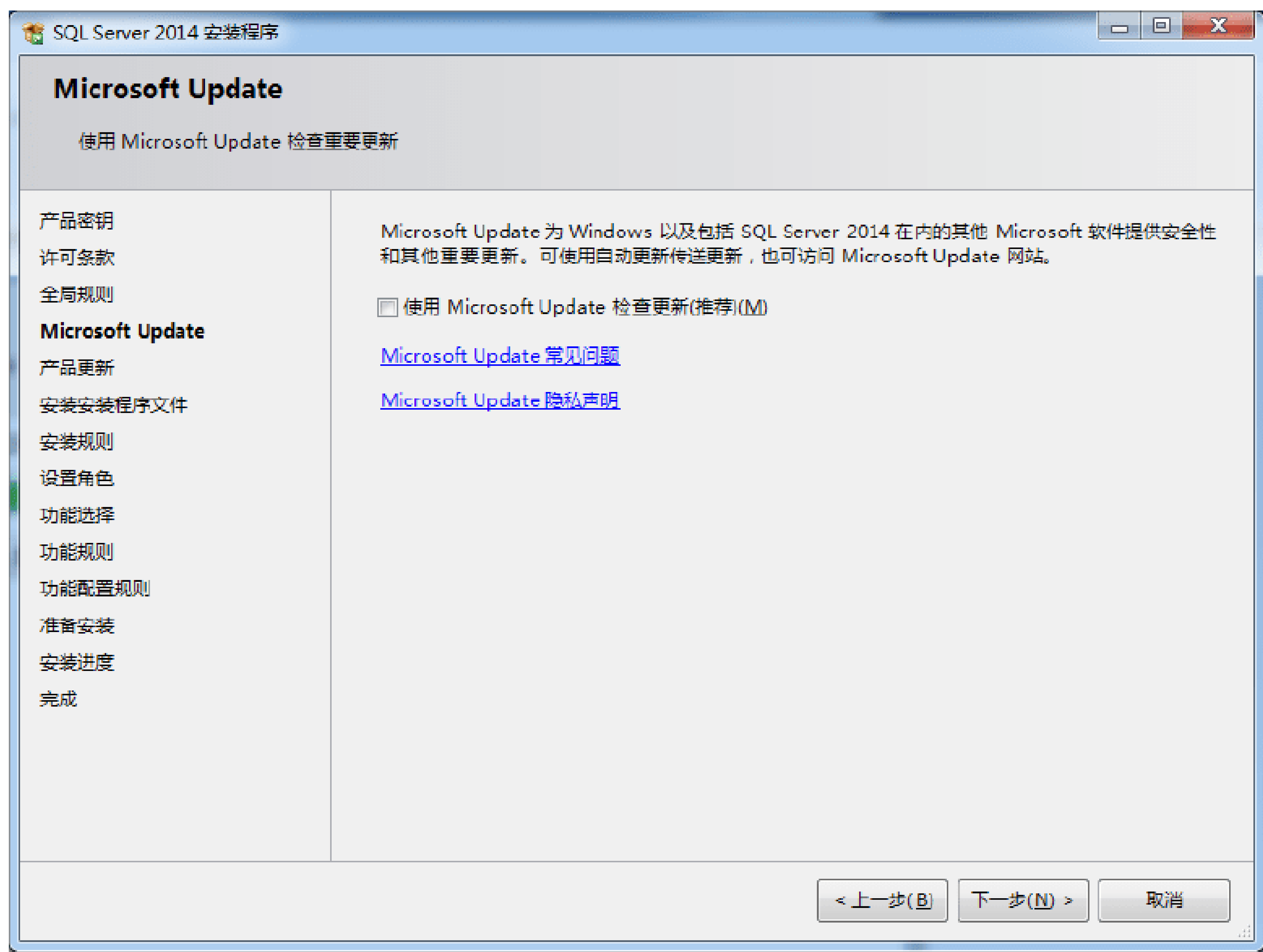


图 2.5　Microsoft Update 界面

（6）单击“下一步”按钮，进入“产品更新”界面，该界面中出现的错误提示是 Windows 系统没有设置自动更新，不用理会该错误，如图 2.6 所示。

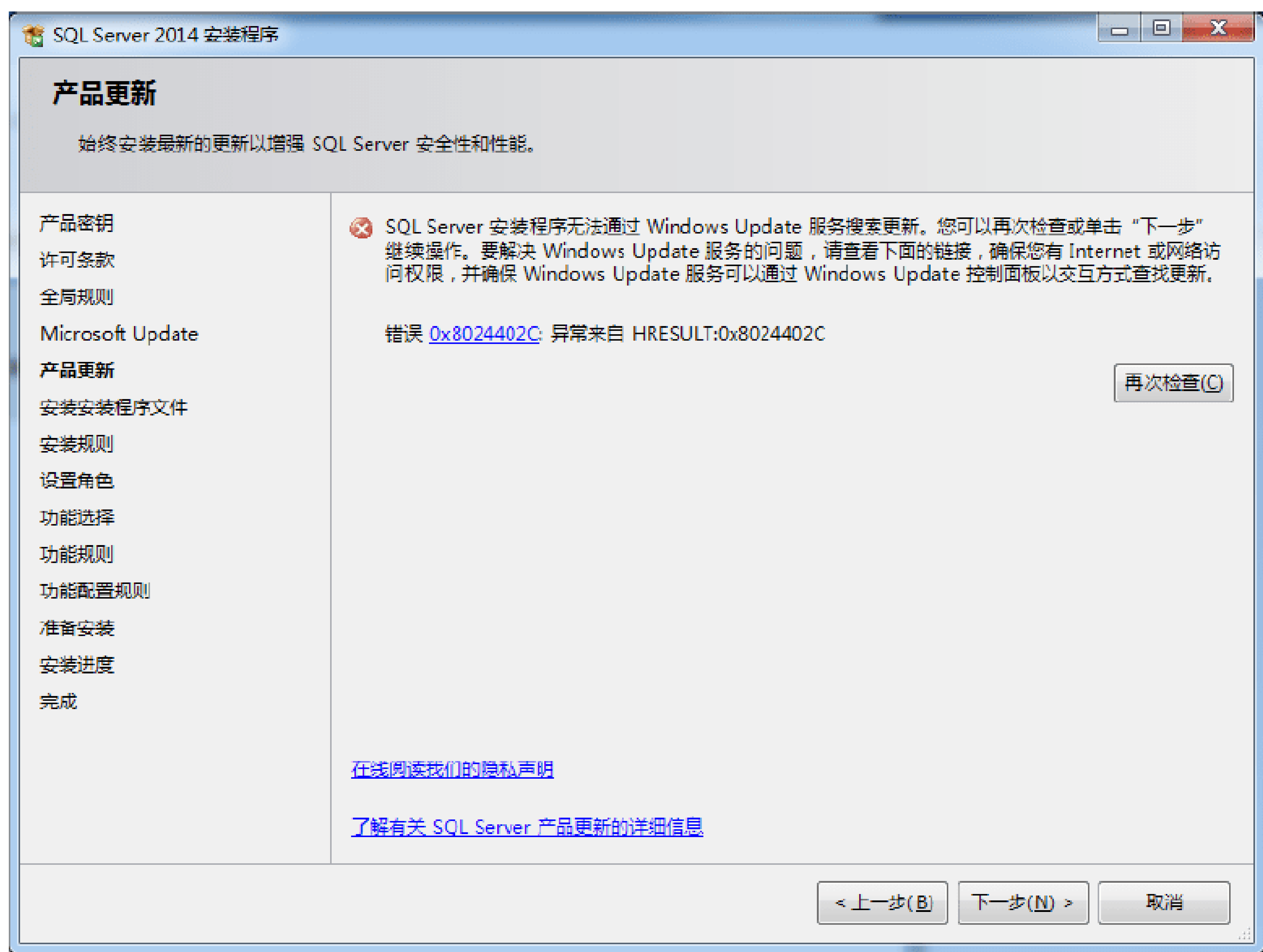


图 2.6　“产品更新”界面

（7）单击“下一步”按钮，进入“安装安装程序文件”界面，如图 2.7 所示，该界面中安装完必要的程序文件后，“下一步”按钮变为可用。



图 2.7　“安装安装程序文件”界面

（8）单击“下一步”按钮，进入“安装规则”界面，如图 2.8 所示，该界面中如果所有规则都通过，则“下一步”按钮可用。

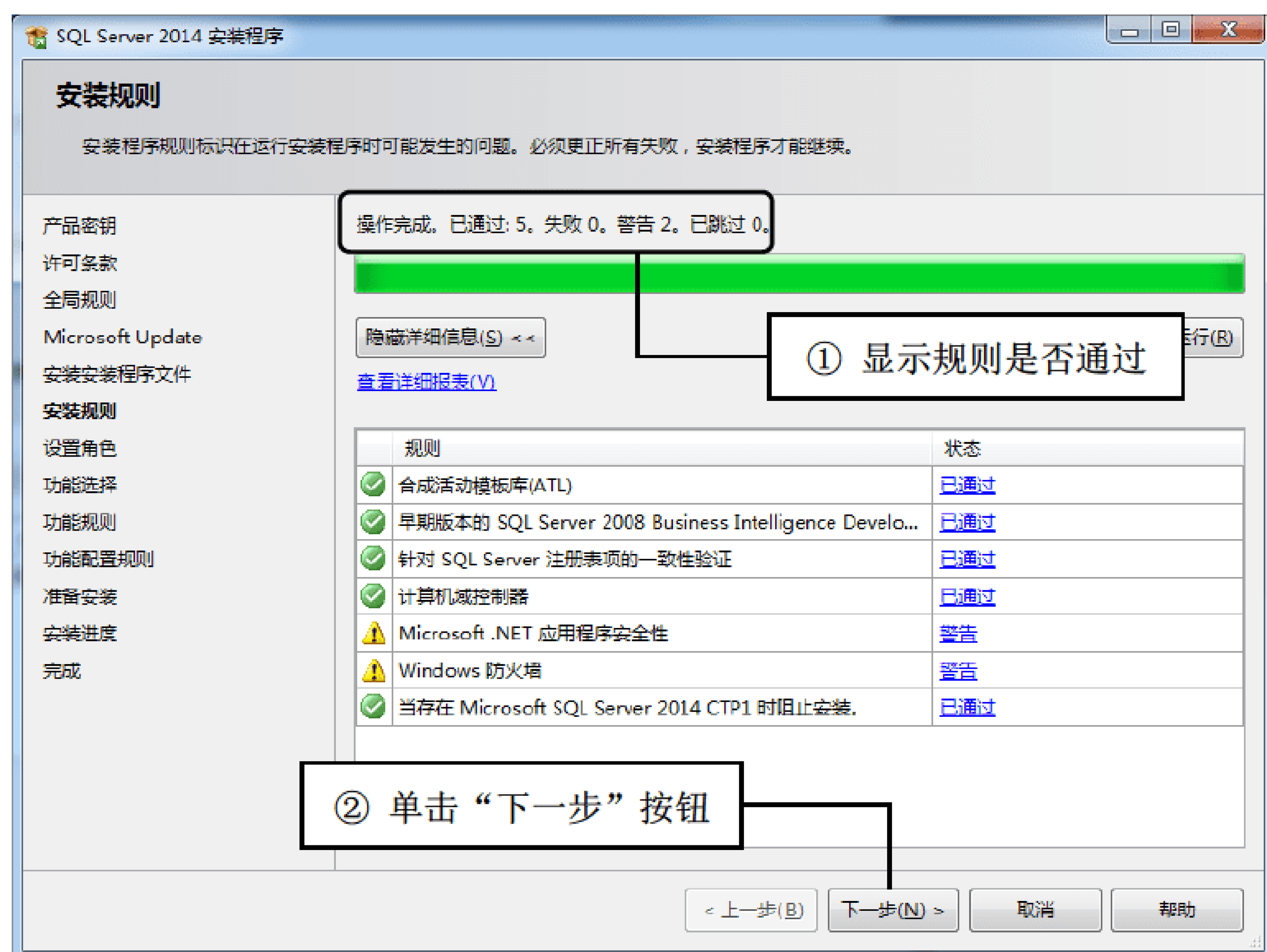


图 2.8　“安装规则”界面

（9）单击“下一步”按钮，进入“设置角色”界面，如图 2.9 所示，选中“SQL Server 功能安装”单选按钮。



图 2.9 “设置角色”界面

（10）单击“下一步”按钮，进入“功能选择”界面，这里可以选择要安装的功能，单击“全选”按钮，选择安装所有功能，如图 2.10 所示。

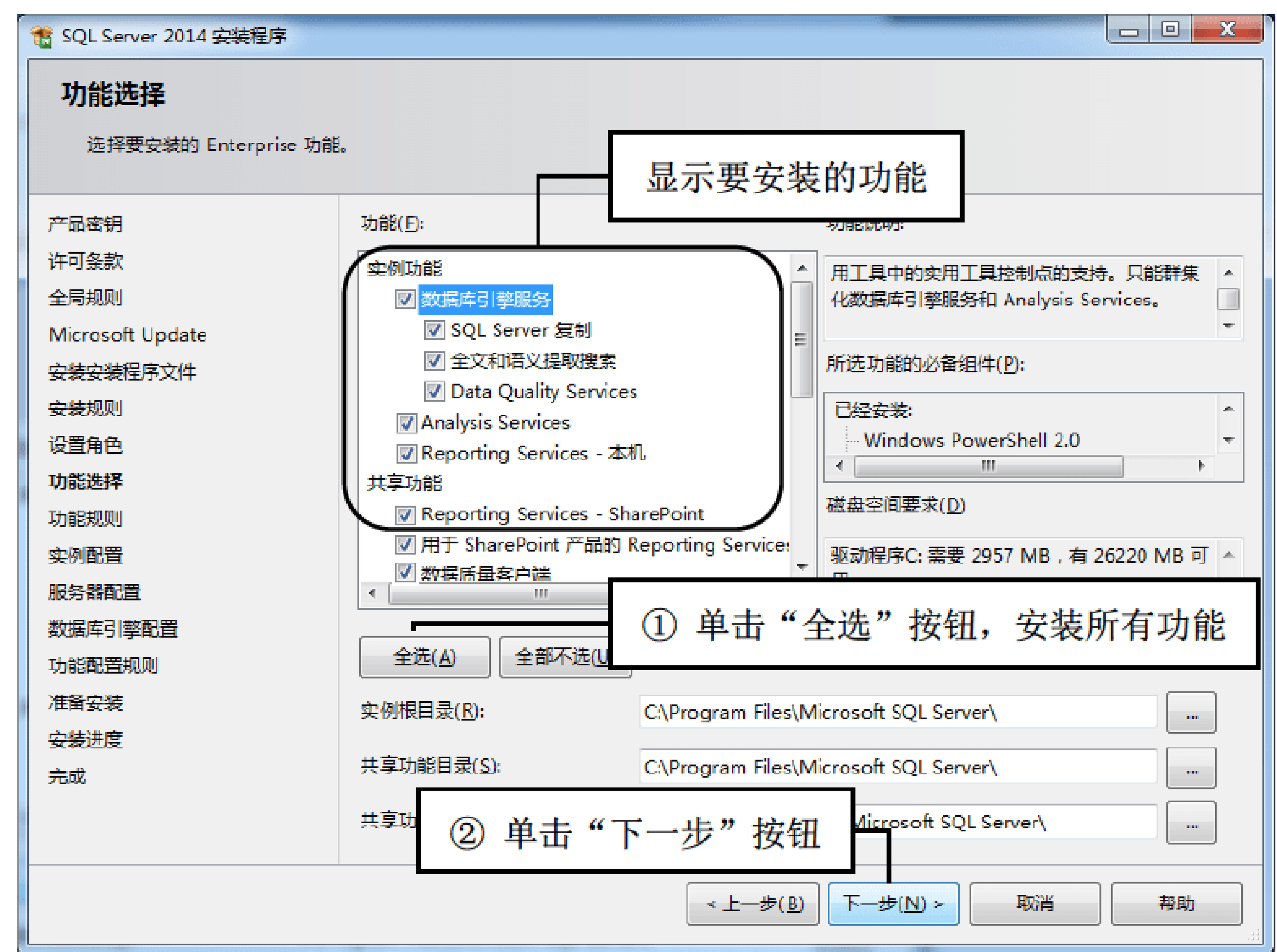


图 2.10 “功能选择”界面

（11）单击“下一步”按钮，进入“实例配置”界面，在该界面中选择实例的命名方式并命名实例，然后选择实例根目录，如图 2.11 所示。



图 2.11　“实例配置”界面

（12）单击“下一步”按钮，进入“服务器配置”界面，如图 2.12 所示。

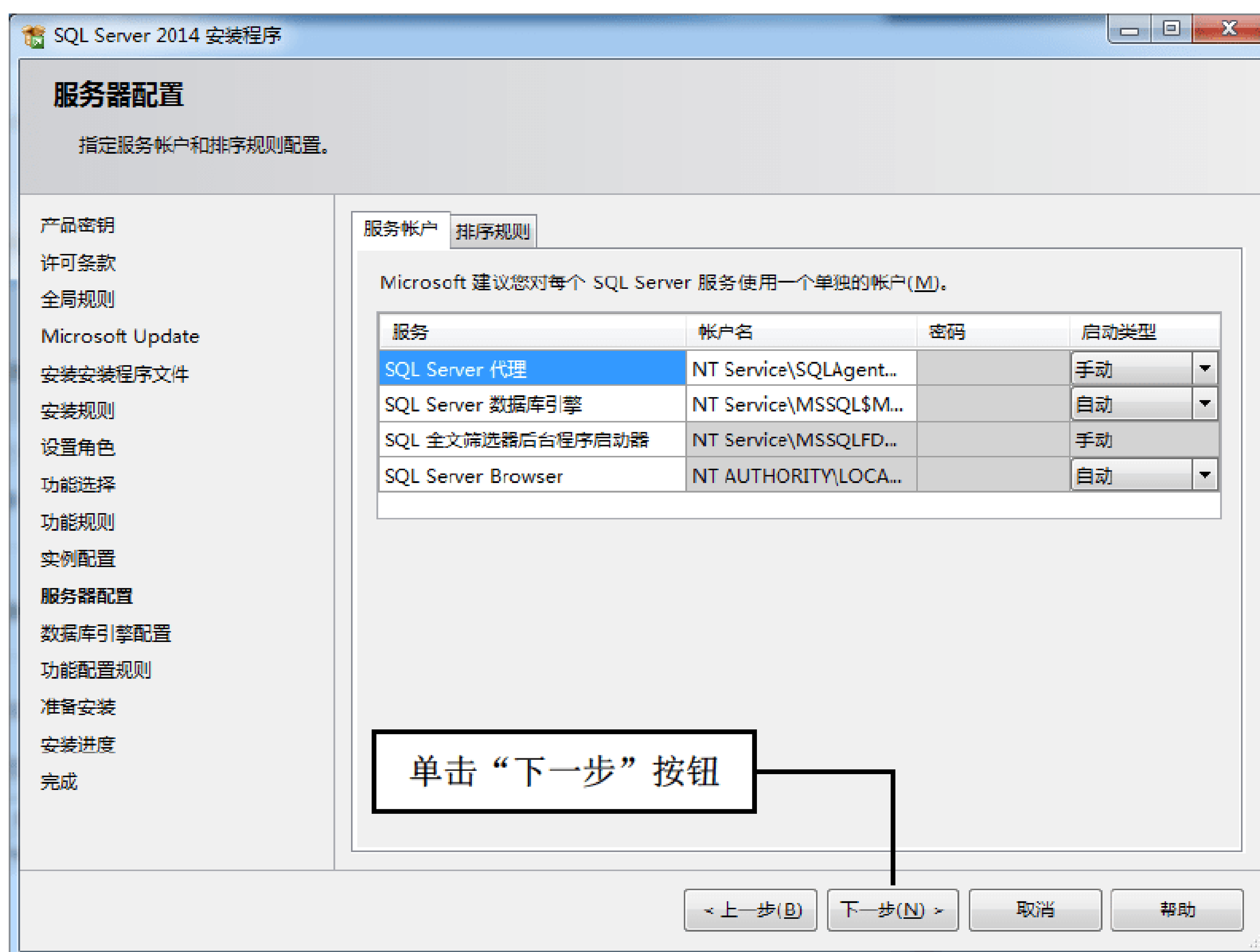


图 2.12　“服务器配置”界面

（13）单击“下一步”按钮，进入“数据库引擎配置”界面，该界面中选择身份验证模式，并输入密码；然后单击“添加当前用户”按钮，如图 2.13 所示。



图 2.13 “数据库引擎配置”界面

（14）单击“下一步”按钮，进入“准备安装”界面，如图 2.14 所示，该界面中显示准备安装的 SQL Server 2014 功能。



图 2.14 “准备安装”界面

表 1.1　T-SQL 语法规则

约　　定	说　　明
UPPERCASE（大写）	T-SQL 关键字
Italic（斜体）	用户提供的 T-SQL 语法的参数
Bold（粗体）	数据库名、表名、列名、索引名、存储过程、实用工具、数据类型名以及必须按所显示的原样键入的文本
下画线	指示当语句中省略了包含带下画线的值的子句时应用的默认值
|（竖线）	分隔括号或大括号中的语法项。只能选择其中一项
[ ]（方括号）	可选语法项。不要输入方括号
{ }（大括号）	必选语法项。不要输入大括号
[,...n]	指示前面的项可以重复 n 次。每一项由逗号分隔
[...n]	指示前面的项可以重复 n 次。每一项由空格分隔
[;]	可选的 T-SQL 语句终止符。不要输入方括号
<label> :: =	语法块的名称。此约定用于对可在语句中的多个位置使用的过长语法段或语法单元进行分组和标记。可使用的语法块的每个位置应括在尖括号内

2. Transact-SQL 语言分类

Transact-SQL 语言的分类如下。

（1）变量说明语句：用来说明变量的命令。

（2）数据定义语言：用来建立数据库、数据库对象和定义列，大部分是以 CREATE 开头的命令，如 CREATE TABLE、CREATE VIEW 和 DROP TABLE 等。

（3）数据操纵语言：用来操纵数据库中数据的命令，如 SELECT、INSERT、UPDATE、DELETE 和 CURSOR 等。

（4）数据控制语言：用来控制数据库组件的存取许可、存取权限等命令。

（5）流程控制语言：用于设计应用程序流程的语句，如 IF WHILE 和 CASE 等。

（6）内嵌函数：实现参数化视图的功能。

（7）其他命令：嵌于命令中使用的标准函数。

1.6　小　　结

本章介绍了数据库的基本概念：数据库系统的组成、数据库三级模式结构及映射、关系数据库的规范化及设计原则等。通过本章的学习，读者可以对数据库有一个系统的了解，在此基础上了解 Transact-SQL 语言，为进一步的学习奠定基础。

第 2 章

SQL Server 2014 安装与配置

（视频讲解：11 分钟）

本章内容包括 SQL Server 2014 简介、安装 SQL Server 2014、启动 SQL Server 2014 管理工具、脚本与批处理，以及数据库的备份和还原、分离和附加、导入和导出。通过本章的学习，读者应该熟悉 SQL Server 2014，选择合适的版本进行安装和配置，并掌握操作 SQL Server 2014 数据库的方法等。

学习摘要：

- **SQL Server 简介**
- **安装 SQL Server 2014**
- **脚本与批处理**
- **备份和还原数据库**
- **分离和附加数据库**
- **导入和导出数据库或数据表**

2.1　SQL Server 数据库简介

SQL Server 是由微软（Microsoft）公司开发的一个大型的关系数据库系统，它为用户提供了一个安全、可靠、易管理和高端的客户/服务器数据库平台。

SQL Server 数据库的中心数据驻留在一个中心计算机上，该计算机被称为服务器。用户通过客户机的应用程序来访问服务器上的数据库，在被允许访问数据库之前，SQL Server 首先对来访问的用户请求做安全验证，只有验证通过后才能够进行处理请求，并将处理的结果返回给客户机应用程序。


视频讲解

2.2　安装 SQL Server

SQL Server 是微软公司推出的数据库服务器工具，从最初的 SQL Server 2000 版本起步，逐渐发展到至今的 SQL Server 2017，深受广大开发者的喜爱。从 SQL Server 2005 版本之后，SQL Server 数据库的安装与配置过程类似，这里以 SQL Server 2014 版本为例讲解 SQL Server 数据库的安装与配置过程。

2.2.1　SQL Server 2014 安装必备

安装 SQL Server 2014 之前，首先要了解安装所需的必备条件，检查计算机的软硬件配置是否满足 SQL Server 2014 的安装要求，具体要求如表 2.1 所示。

表 2.1　安装 SQL Server 2014 所需的必备条件

名　称	说　明
操作系统	Windows 7（SP1）、Windows 8、Windows 8.1、Windows Server 2008 R2 SP1（x64）、Windows Server 2012（x64）、Windows 10
软件	SQL Server 安装程序需要使用 Microsoft Windows Installer 4.5 或更高版本以及 Microsoft 数据访问组件（MDAC）2.8 SP1 或更高版本
处理器	1.4GHz 处理器，建议使用 2.0GHz 或速度更快的处理器
内存	最小 2GB，建议使用 4GB 或更大的内存
可用硬盘空间	至少 2.2GB 的可用硬盘空间
驱动器	从磁盘进行安装时需要相应的 DVD 驱动器
显示器	SQL Server 2014 要求有 Super-VGA（800×600）或更高分辨率的显示器

2.2.2　SQL Server 2014 的安装

安装 SQL Server 2014 数据库的步骤如下。

（1）使用虚拟光驱软件加载下载的 SQL Server 2014 的安装镜像文件（.iso 文件），在“SQL Server

安装中心”窗口中单击左侧的“安装”选项，再单击“全新 SQL Server 独立安装或向现有安装添加功能”超链接，如图 2.1 所示。

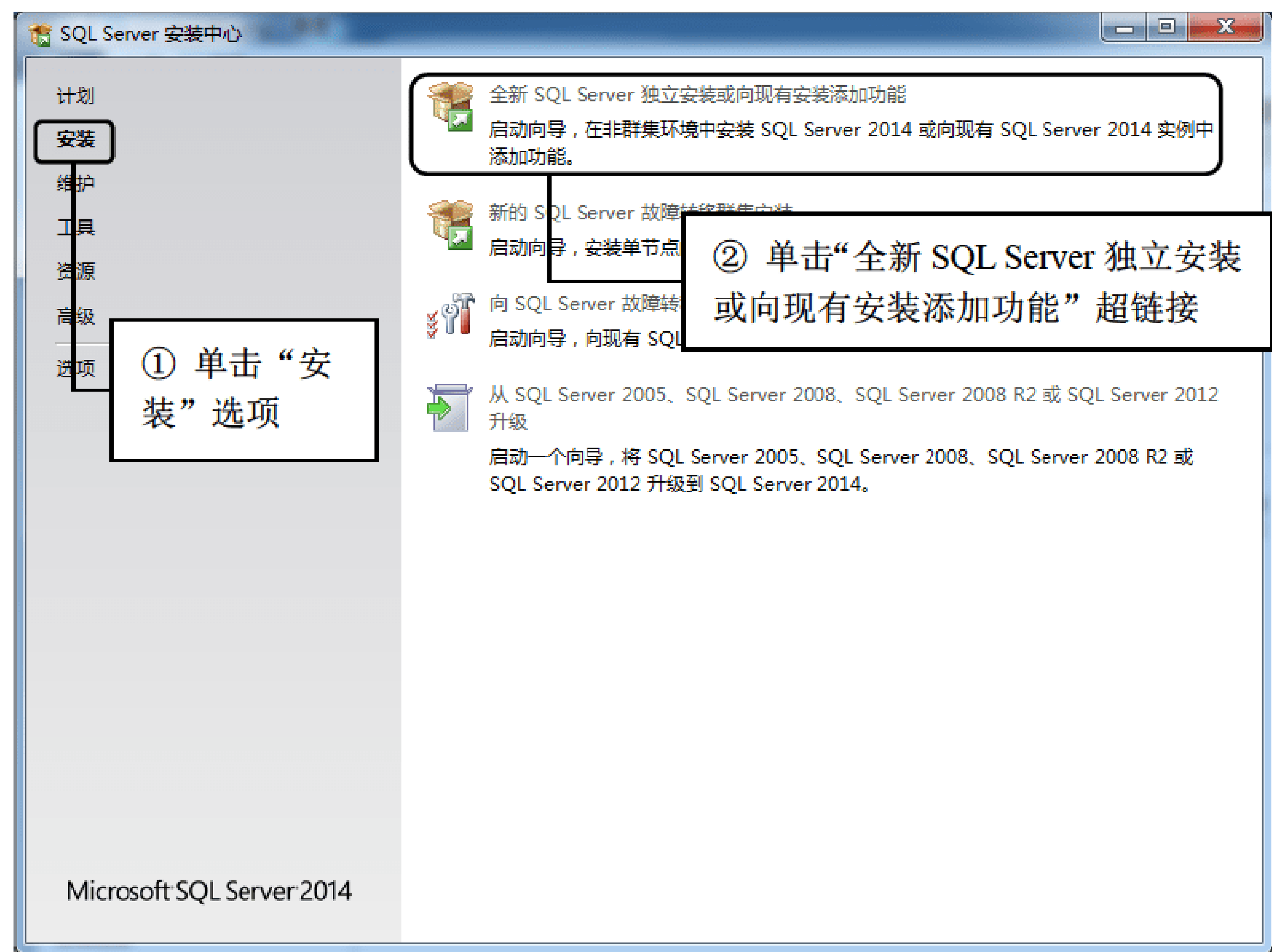


图 2.1　单击左侧的“安装”选项

（2）单击“下一步”按钮，打开“产品密钥”界面，如图 2.2 所示，该界面中输入产品密钥。

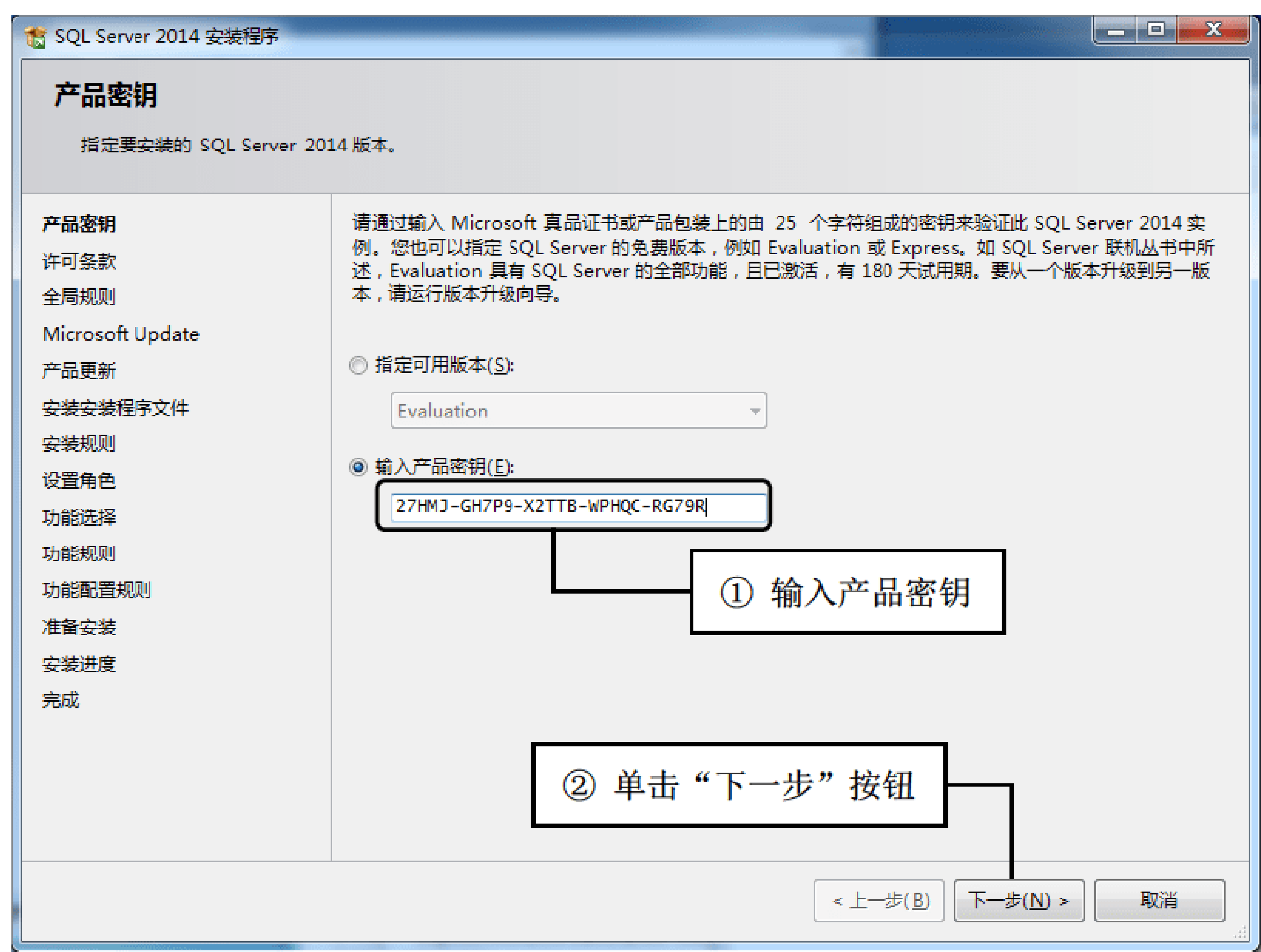


图 2.2　“产品密钥”界面

（3）单击“下一步”按钮，进入“许可条款”界面，如图 2.3 所示，选中“我接受许可条款”复选框。

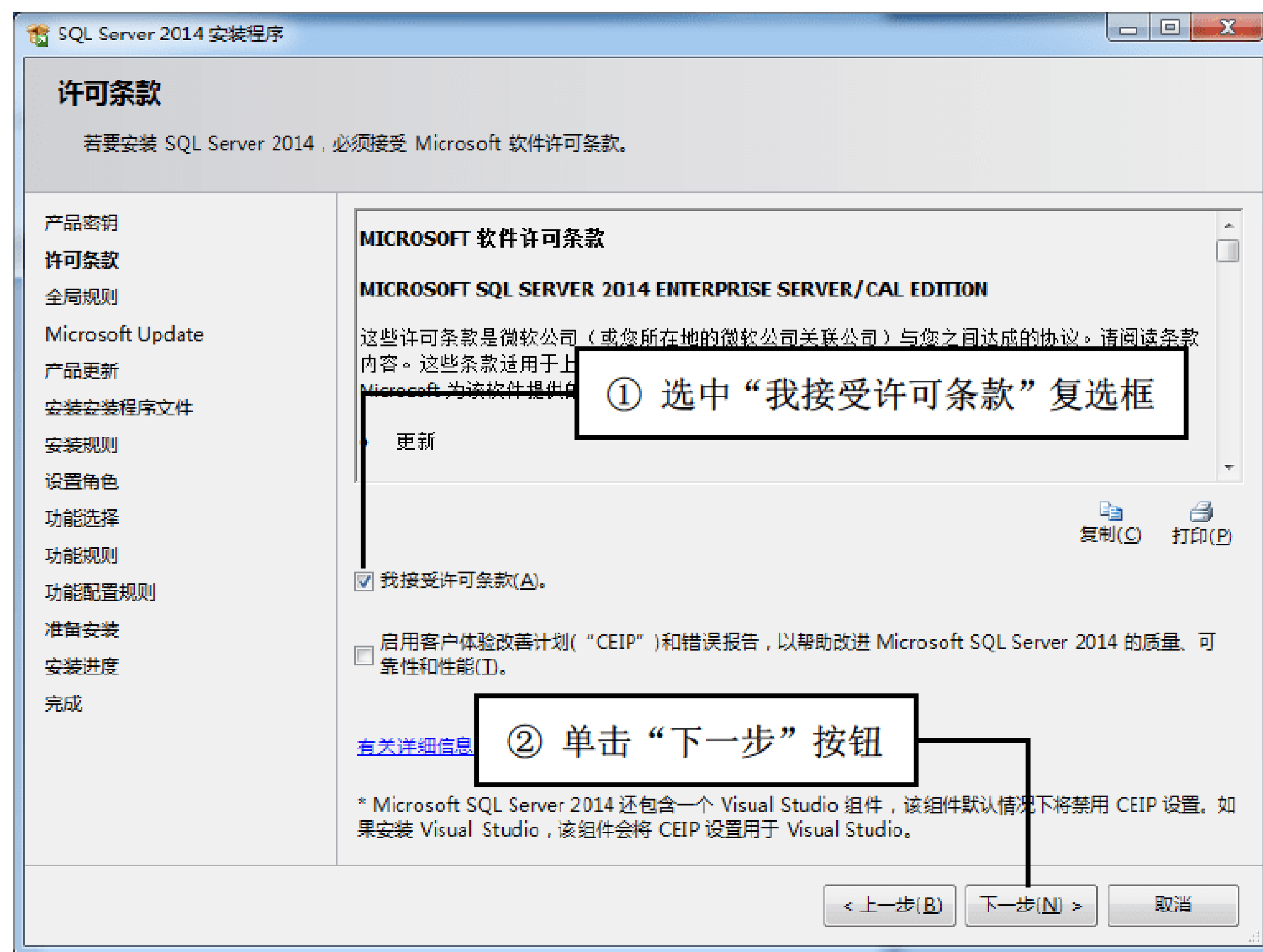


图 2.3　“许可条款”界面

（4）单击“下一步”按钮，进入“全局规则”界面，规则检查完成后，“下一步”按钮可用，如图 2.4 所示。



图 2.4　“全局规则”界面

（5）单击“下一步”按钮，进入 Microsoft Update 界面，如图 2.5 所示。

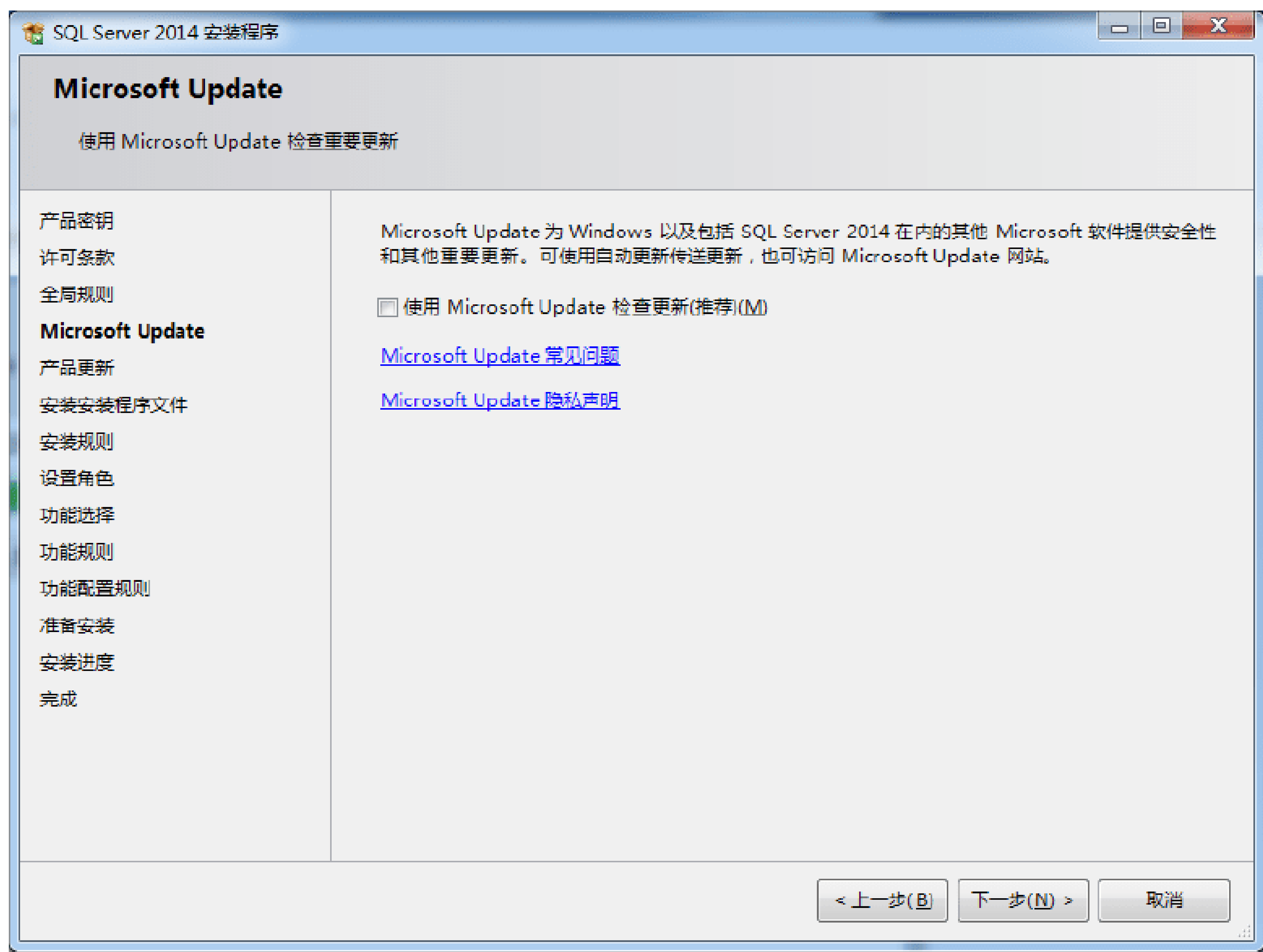


图 2.5　Microsoft Update 界面

（6）单击“下一步”按钮，进入“产品更新”界面，该界面中出现的错误提示是 Windows 系统没有设置自动更新，不用理会该错误，如图 2.6 所示。

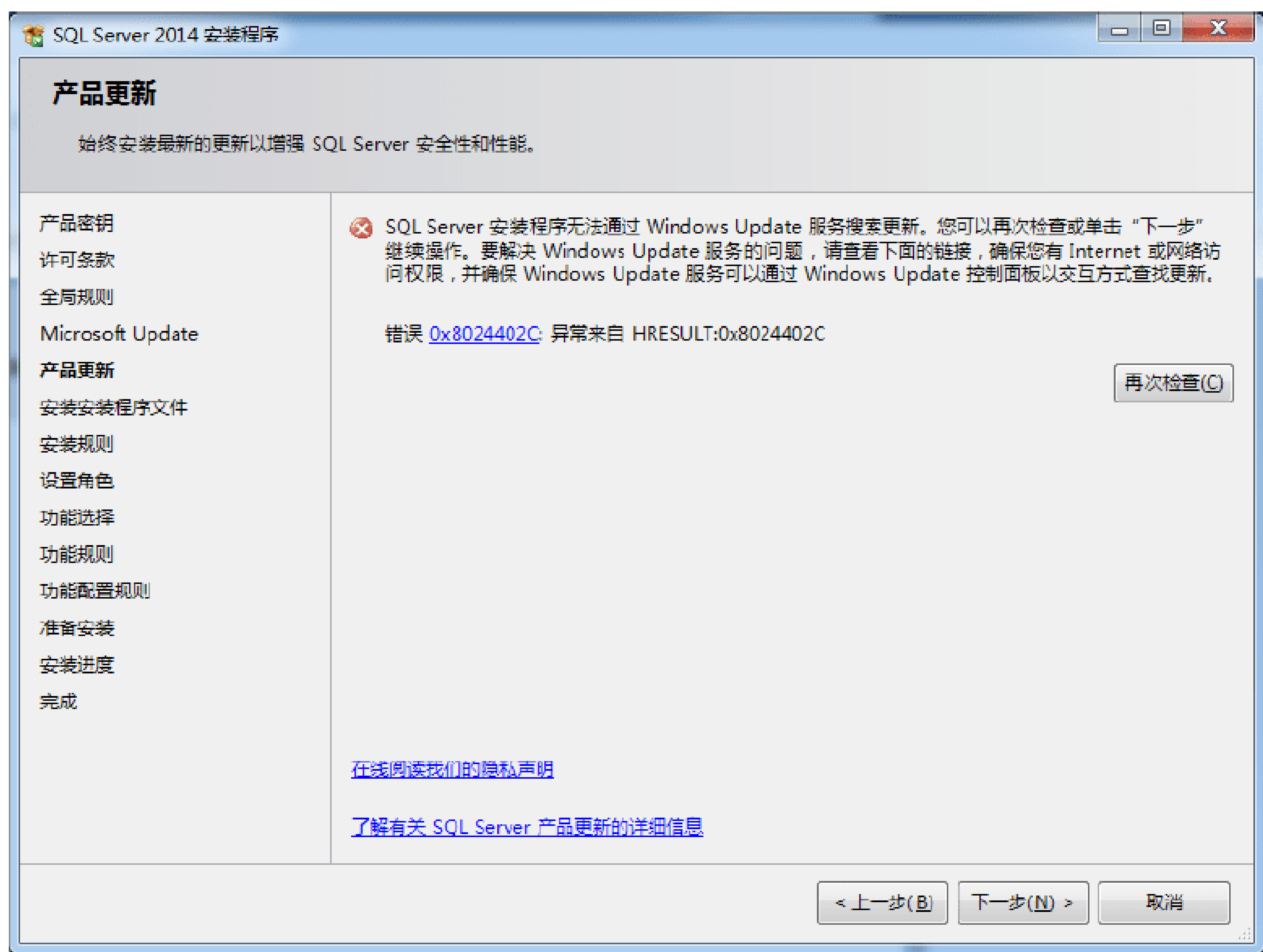


图 2.6　“产品更新”界面

（7）单击“下一步”按钮，进入“安装安装程序文件”界面，如图 2.7 所示，该界面中安装完必要的程序文件后，“下一步”按钮变为可用。



图 2.7　“安装安装程序文件”界面

（8）单击“下一步”按钮，进入“安装规则”界面，如图 2.8 所示，该界面中如果所有规则都通过，则“下一步”按钮可用。

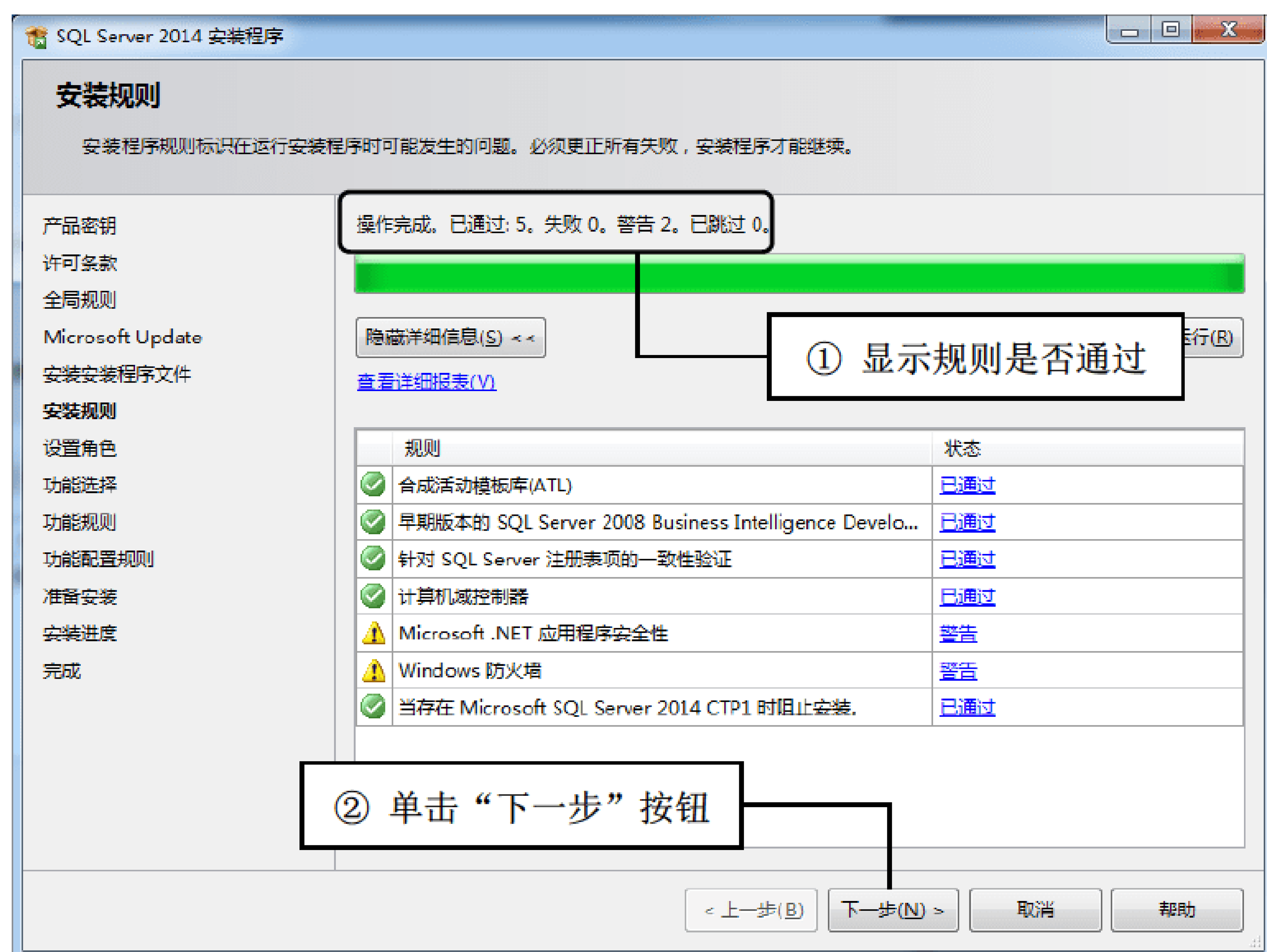


图 2.8　“安装规则”界面

（9）单击“下一步”按钮，进入“设置角色”界面，如图 2.9 所示，选中“SQL Server 功能安装”单选按钮。



图 2.9　“设置角色”界面

（10）单击“下一步”按钮，进入“功能选择”界面，这里可以选择要安装的功能，单击“全选”按钮，选择安装所有功能，如图 2.10 所示。

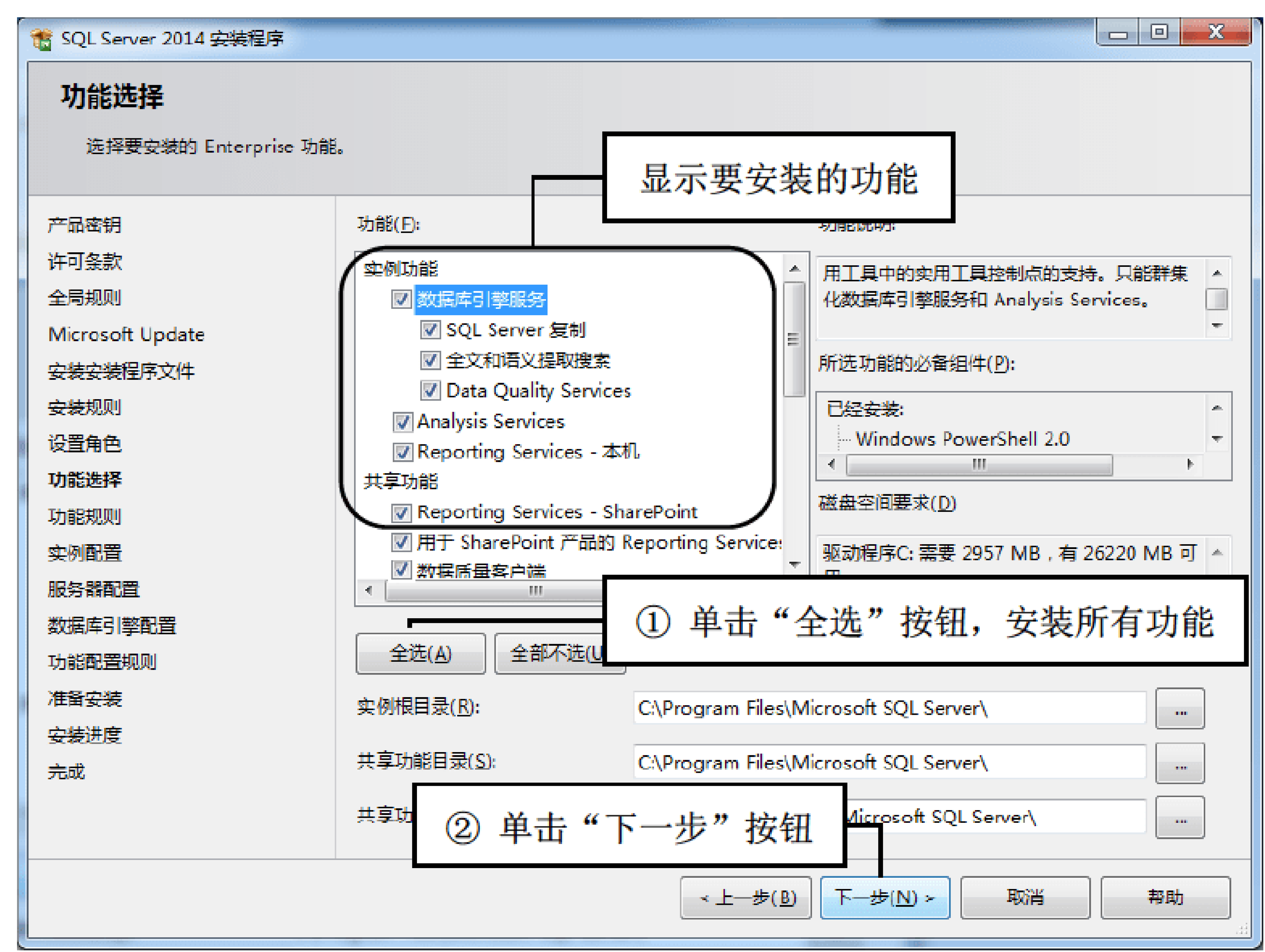


图 2.10　“功能选择”界面

（11）单击“下一步”按钮，进入“实例配置”界面，在该界面中选择实例的命名方式并命名实例，然后选择实例根目录，如图 2.11 所示。



图 2.11　“实例配置”界面

（12）单击“下一步”按钮，进入“服务器配置”界面，如图 2.12 所示。

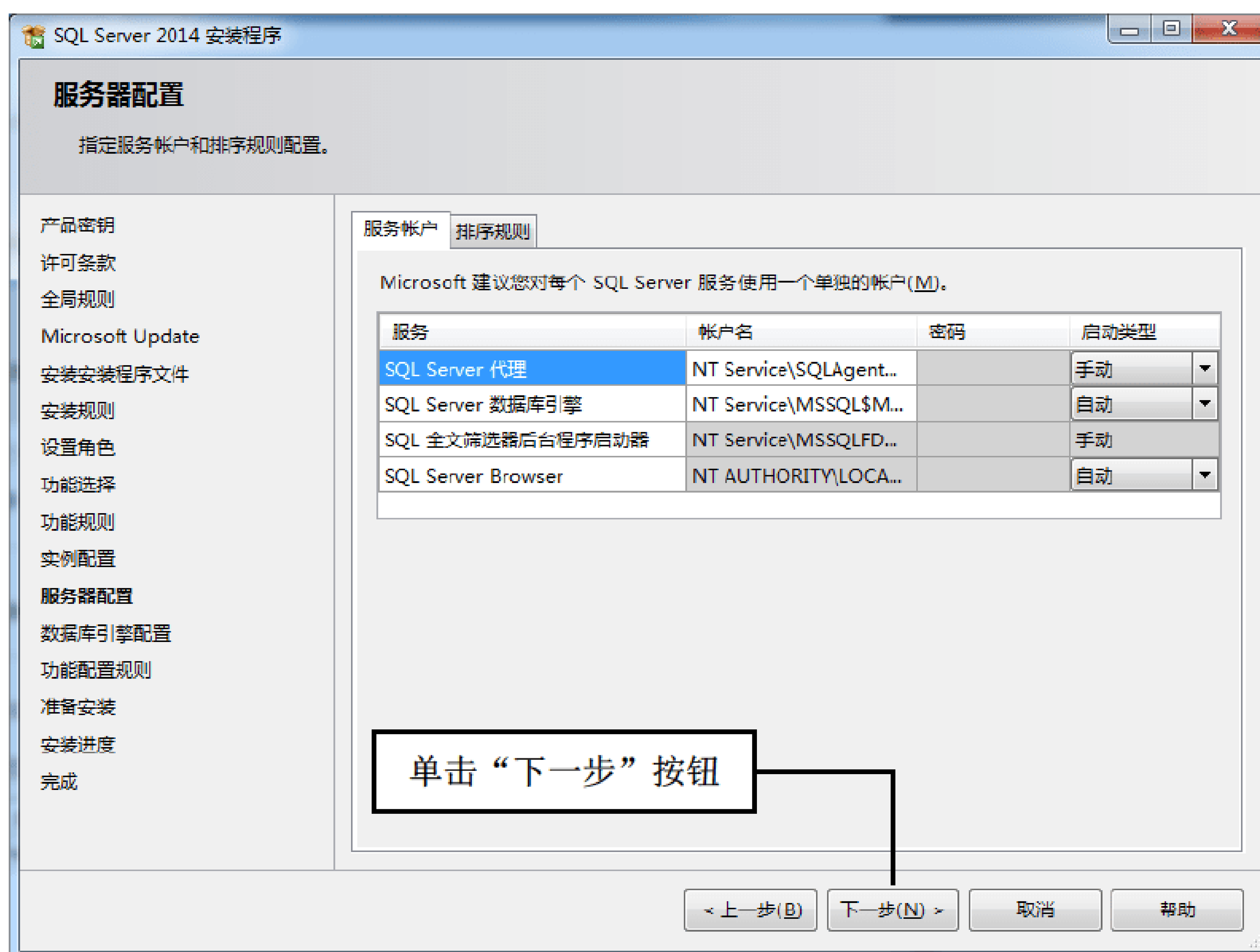


图 2.12　“服务器配置”界面

（13）单击“下一步”按钮，进入“数据库引擎配置”界面，该界面中选择身份验证模式，并输入密码；然后单击“添加当前用户”按钮，如图 2.13 所示。



图 2.13　“数据库引擎配置”界面

（14）单击“下一步”按钮，进入“准备安装”界面，如图 2.14 所示，该界面中显示准备安装的 SQL Server 2014 功能。



图 2.14　“准备安装”界面

（15）单击“安装”按钮，进入“安装进度”界面，如图 2.15 所示，该界面中显示 SQL Server 2014 的安装进度。

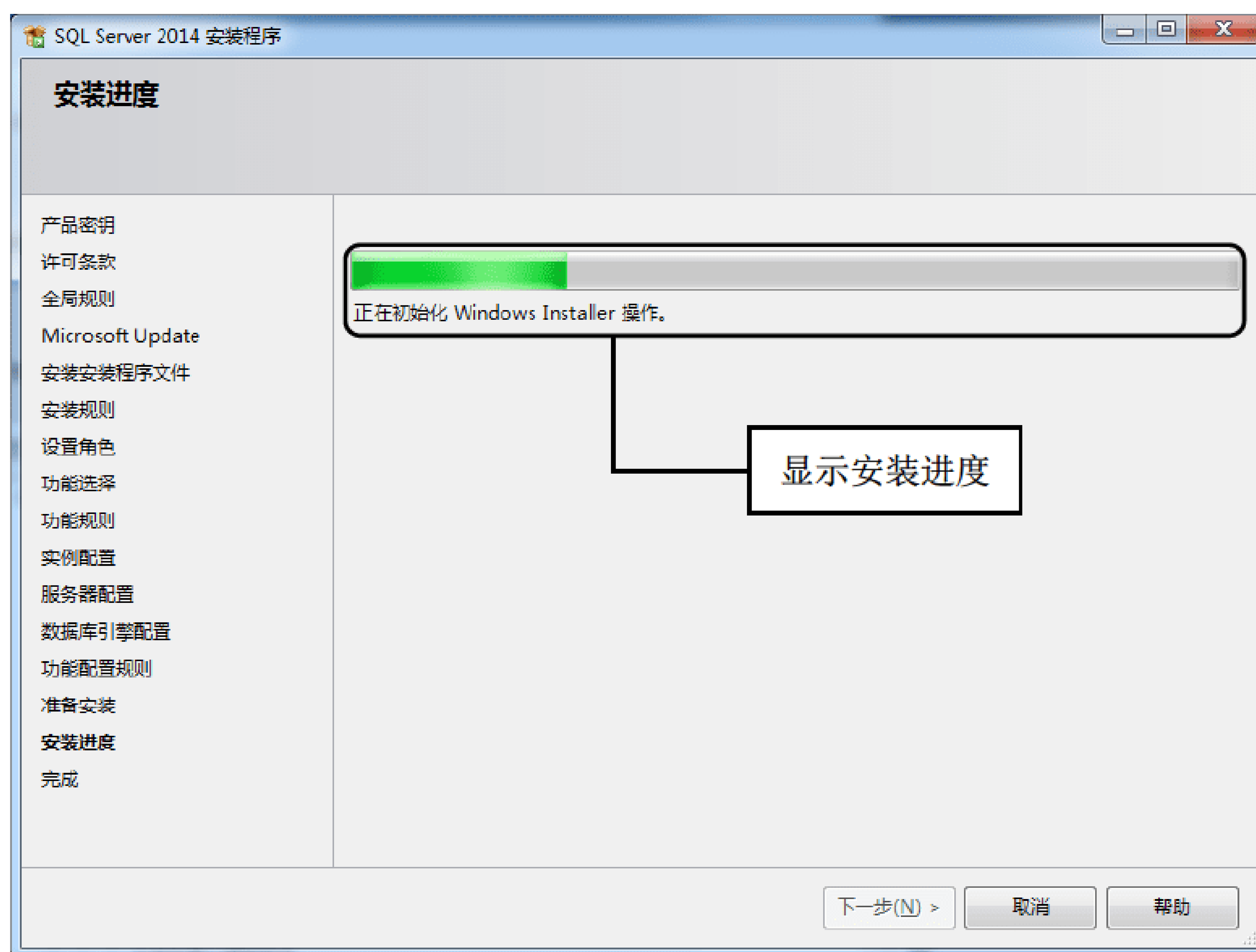


图 2.15　“安装进度”界面

（16）单击“下一步”按钮，进入“完成”界面，如图 2.16 所示，单击“关闭”按钮，即可完成 SQL Server 2014 的安装。

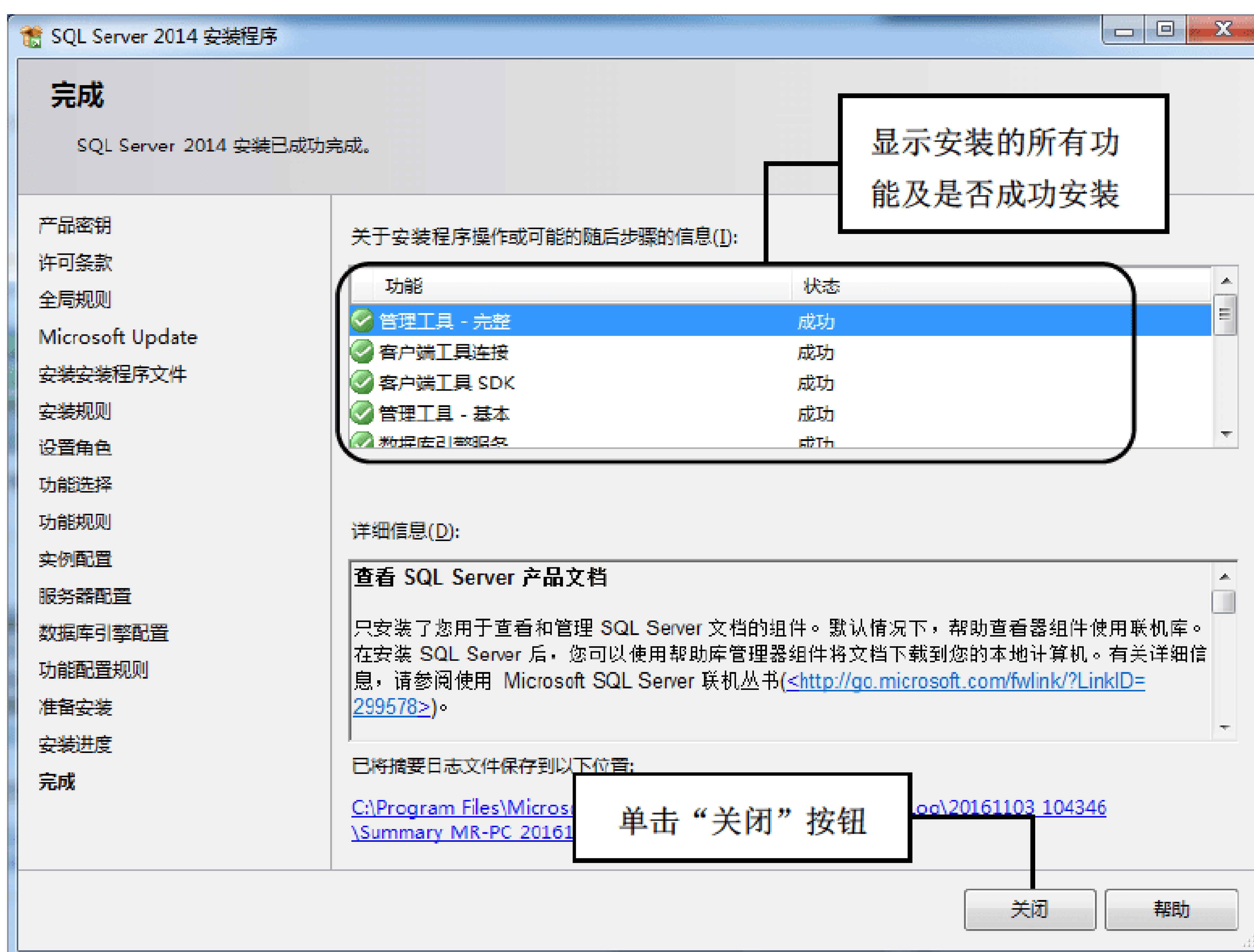


图 2.16　“完成”界面

2.3 启动 SQL Server 2014 管理工具

安装完 SQL Server 2014 后，就可以启动了，具体步骤如下。

（1）选择“开始”→“所有程序”→Microsoft SQL Server 2014→SQL Server 2014 Management Studio 命令，进入“连接到数据库引擎”对话框，如图 2.17 所示。



图 2.17 “连接到数据库引擎”对话框



说明

服务器名称实际上就是安装 SQL Server 2014 设置的实例名称。

（2）在“连接到数据库引擎”对话框中选择自己的服务器名称（通常为默认）和身份验证方式，如果选择的是“Windows 身份验证”，可以直接单击“连接”按钮；如果选择的是“SQL Server 身份验证”，则需要输入在安装 SQL Server 2014 数据库时设置的登录名和密码，其中登录名通常为 sa，密码为用户自己设置，单击“连接”按钮，即可进入 SQL Server Management Studio，如图 2.18 所示。

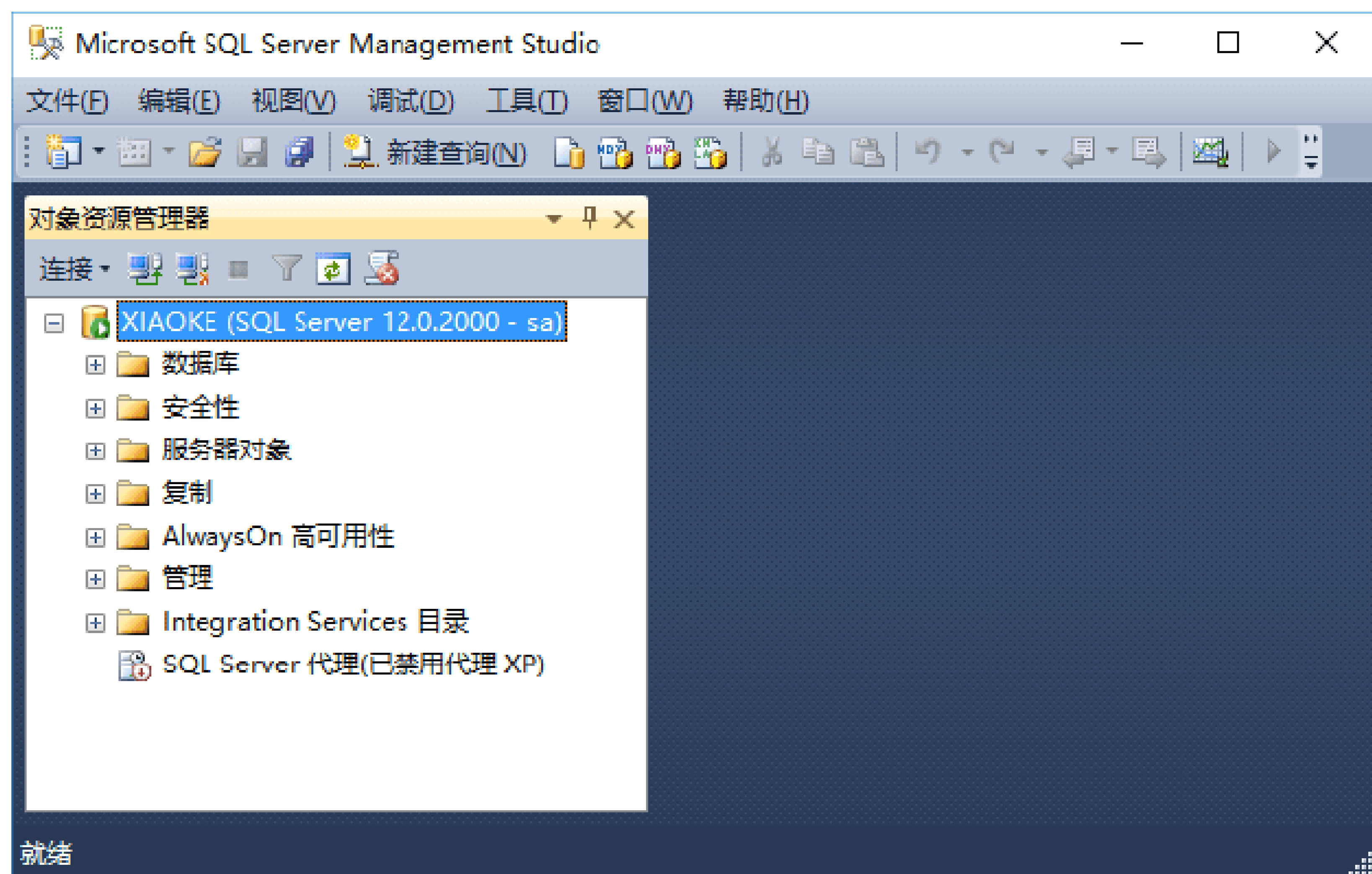


图 2.18 SQL Server Management Studio

2.4　脚本与批处理

2.4.1　将数据库生成脚本

（1）启动 SQL Server Management Studio，在“对象资源管理器”中展开“数据库”节点，选中欲生成脚本的数据库，如图 2.19 所示。

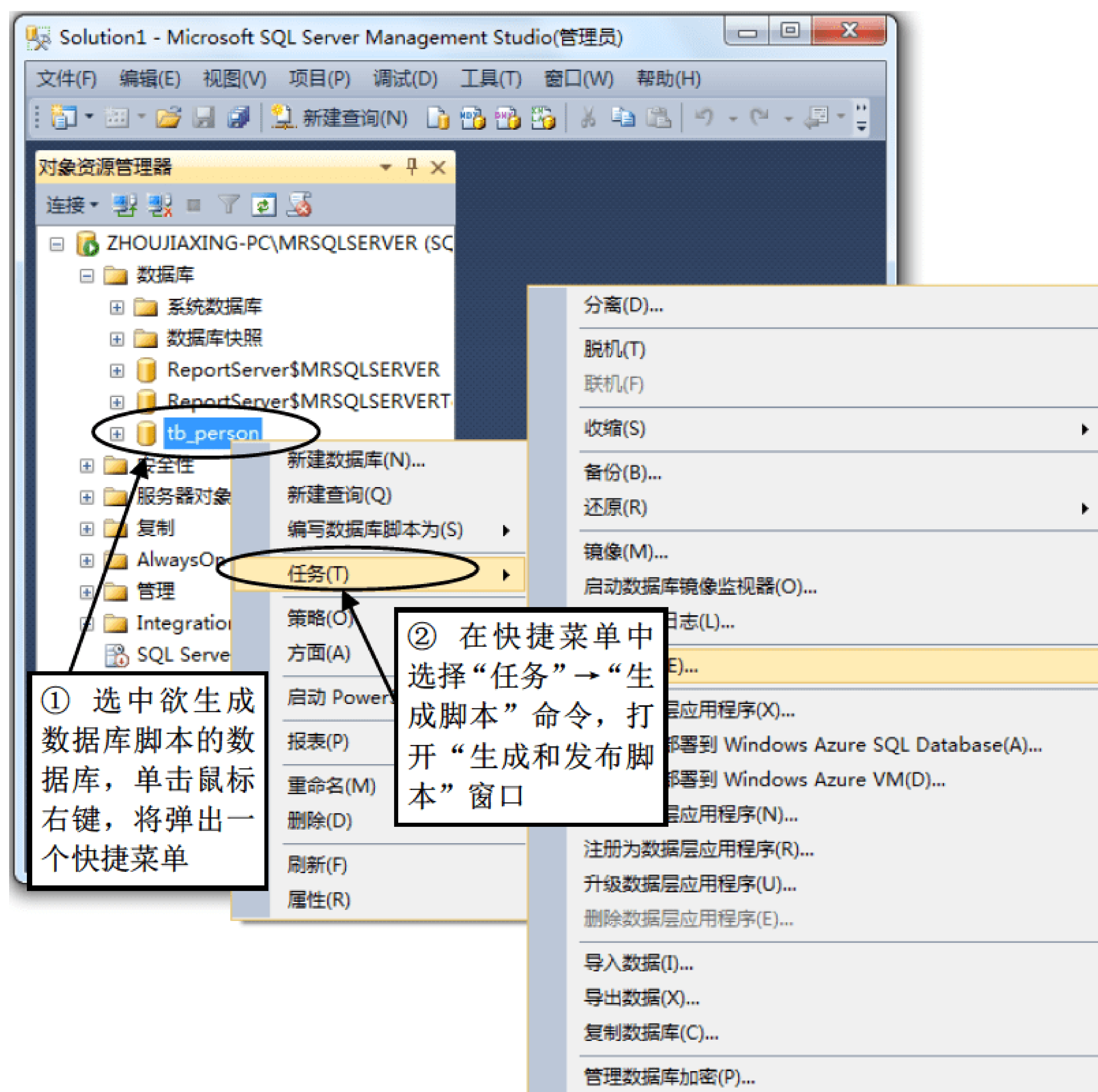


图 2.19　生成脚本

（2）打开“生成和发布脚本”窗口，选中“不再显示此页”复选框，单击“下一步”按钮，如图 2.20 所示。

（3）进入“选择对象”界面，因为要将数据库及数据库中全部数据对象生成脚本，所以直接单击“下一步”按钮，如图 2.21 所示。



图 2.20 “生成和发布脚本”窗口

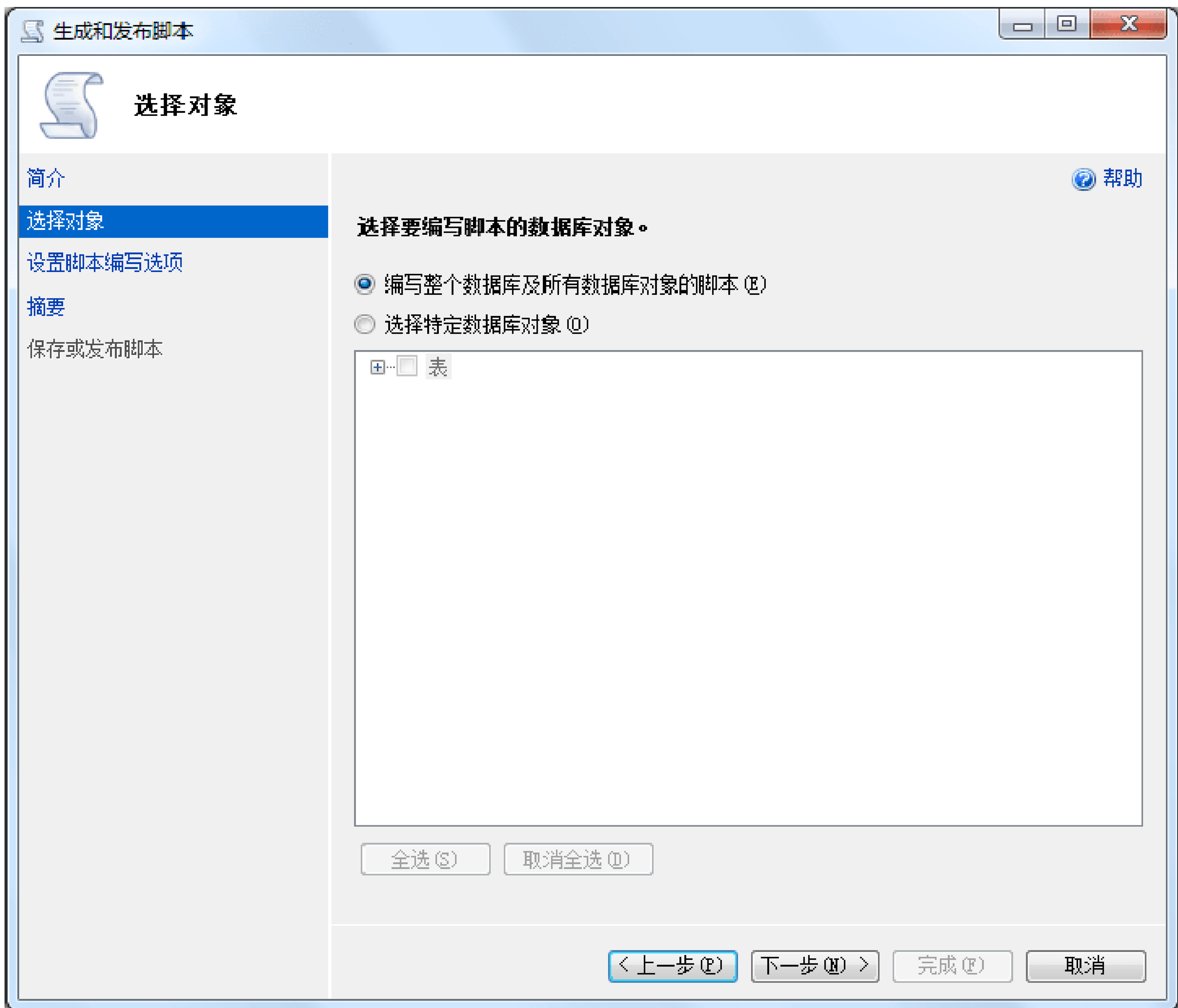


图 2.21 “选择对象”界面

（4）进入“设置脚本编写选项”界面，单击磁盘位置后的...按钮可以选择脚本的保存位置，如图 2.22 所示。

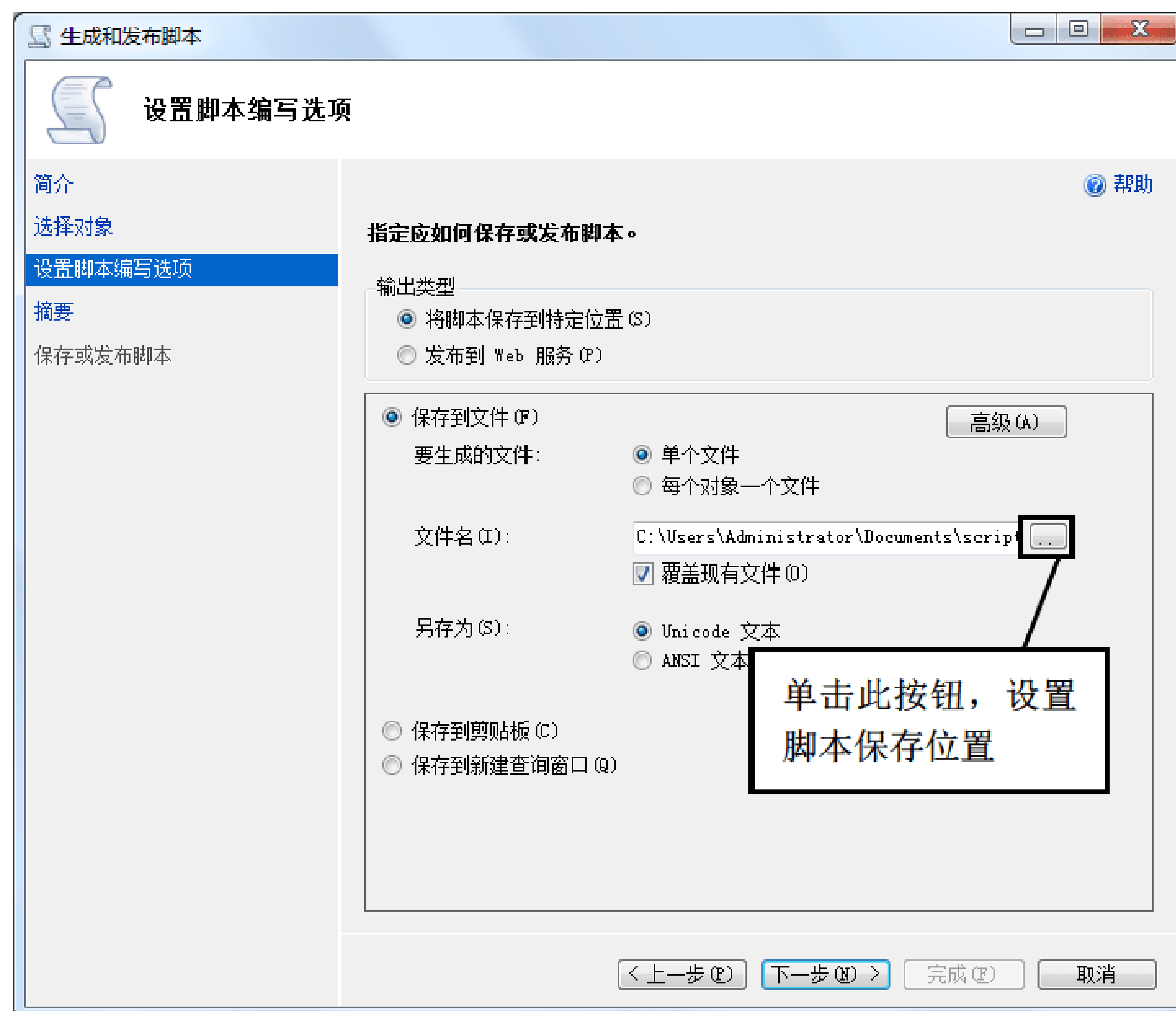


图 2.22　“设置脚本编写选项”界面

（5）选择好脚本保存位置后，单击“下一步”按钮进入“保存或发布脚本”界面，单击“完成”按钮，即可完成数据库脚本文件的生成，如图 2.23 所示。生成的数据库脚本文件如图 2.24 所示。

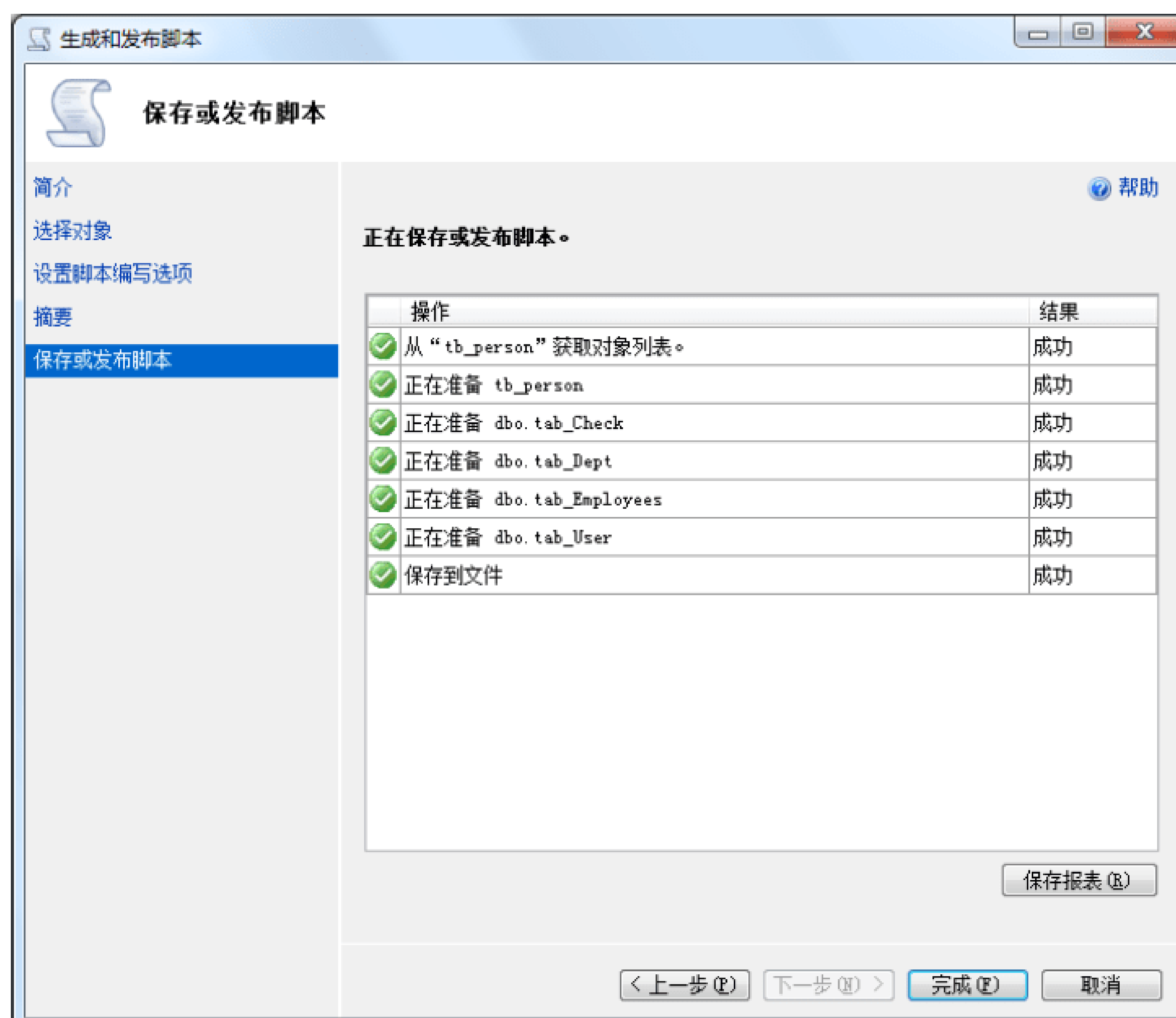


图 2.23　“保存或发布脚本”界面

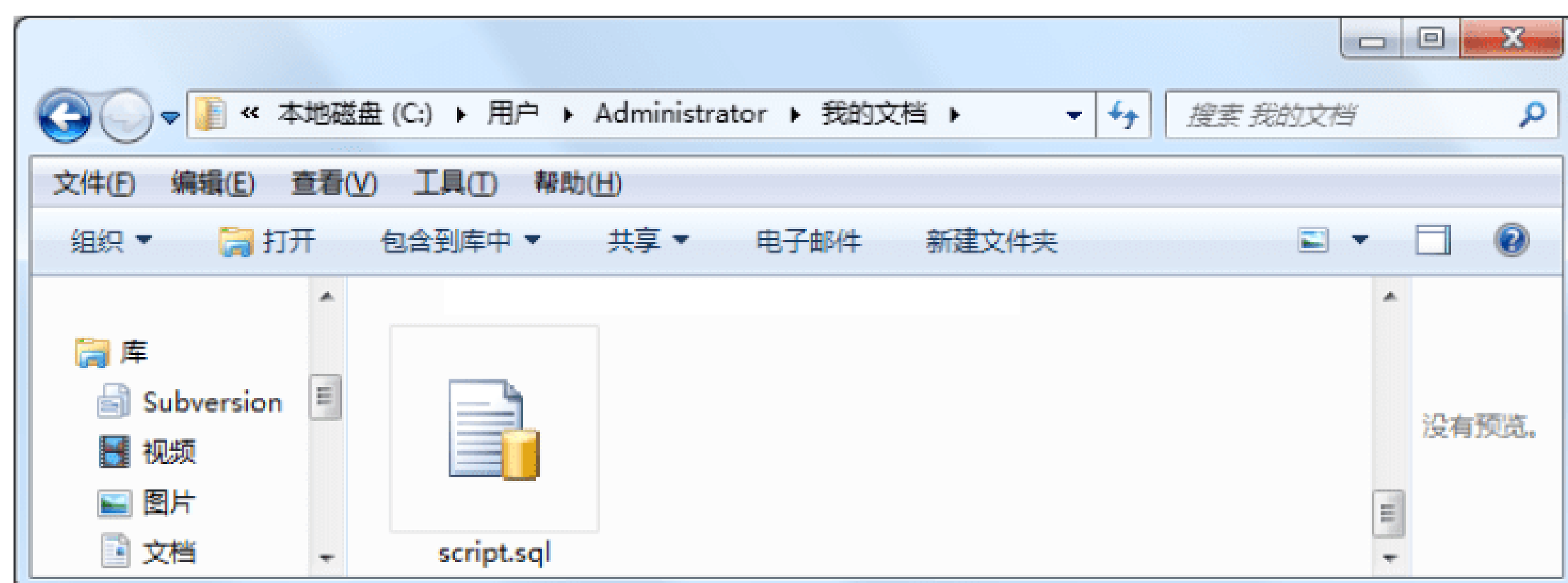


图 2.24　生成的数据库脚本文件

2.4.2　将指定表生成脚本

生成表脚本与生成数据库脚本步骤大致相同，只是在 2.4.1 小节的步骤（3）中选中“选择特定数据库对象”单选按钮，具体步骤请参考 2.4.1 小节。表脚本与数据库脚本的区别在于表脚本会在当前数据库中创建数据表、视图、存储过程等数据库对象，而数据库脚本会创建一个新的数据库，并在该数据库中创建各个数据库对象。

2.4.3　执行脚本

可以在 SQL Server Management Studio 执行脚本。选择“开始”→“所有程序”→Microsoft SQL Server 2014→SQL Server 2014 Management Studio 命令，启动 SQL Server Management Studio，如图 2.25 所示。

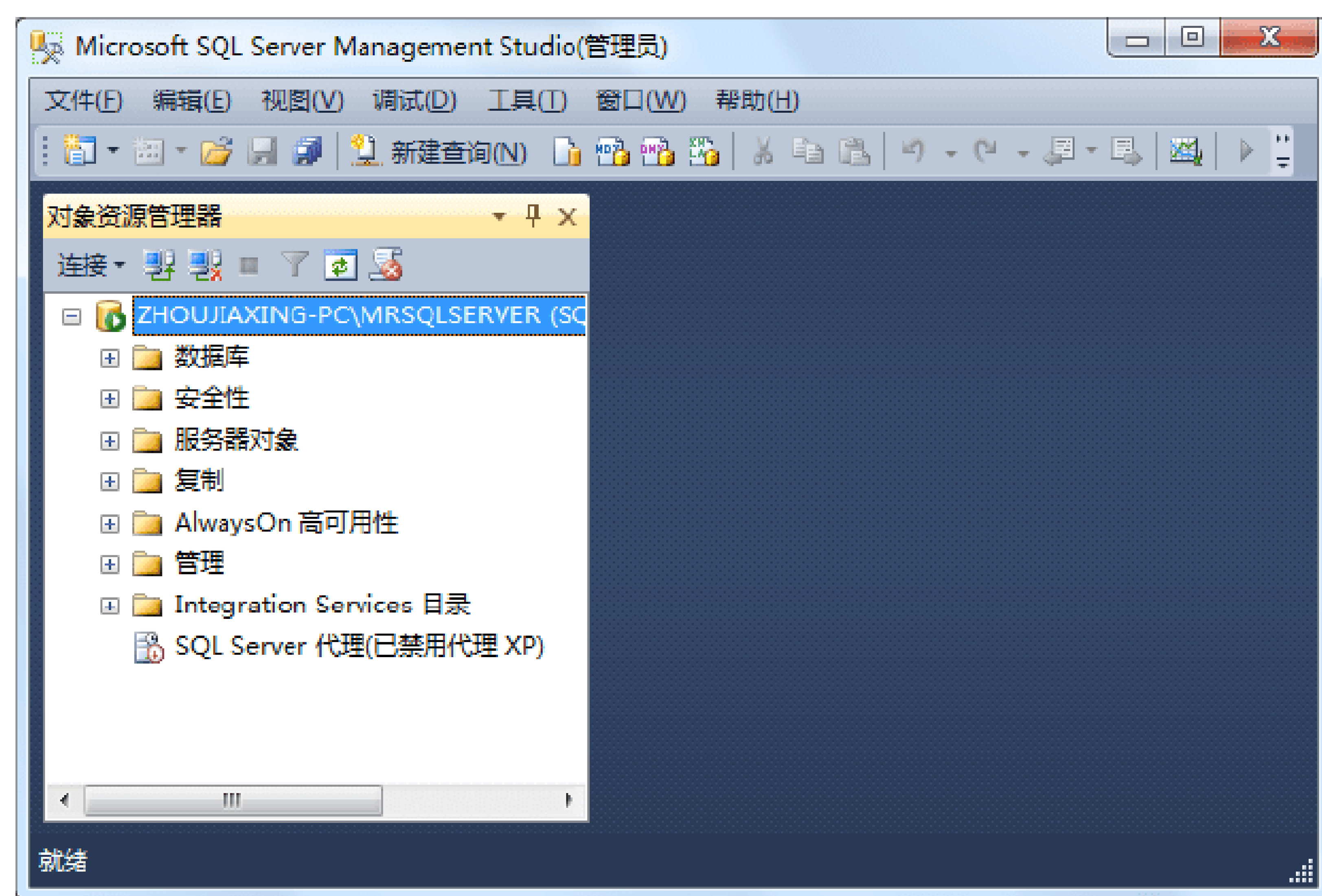


图 2.25　SQL Server Management Studio

在 SQL Server Management Studio 中选择“文件”→“打开”→“文件”命令，打开“打开文件”对话框，如图 2.26 所示。

在“打开文件”对话框中选择需要执行的脚本，如 script.sql，单击“打开”按钮打开脚本，如图 2.27 所示。

在 SQL Server Management Studio 中单击 执行(X) 按钮或按 F5 键执行脚本中的 SQL 语句。

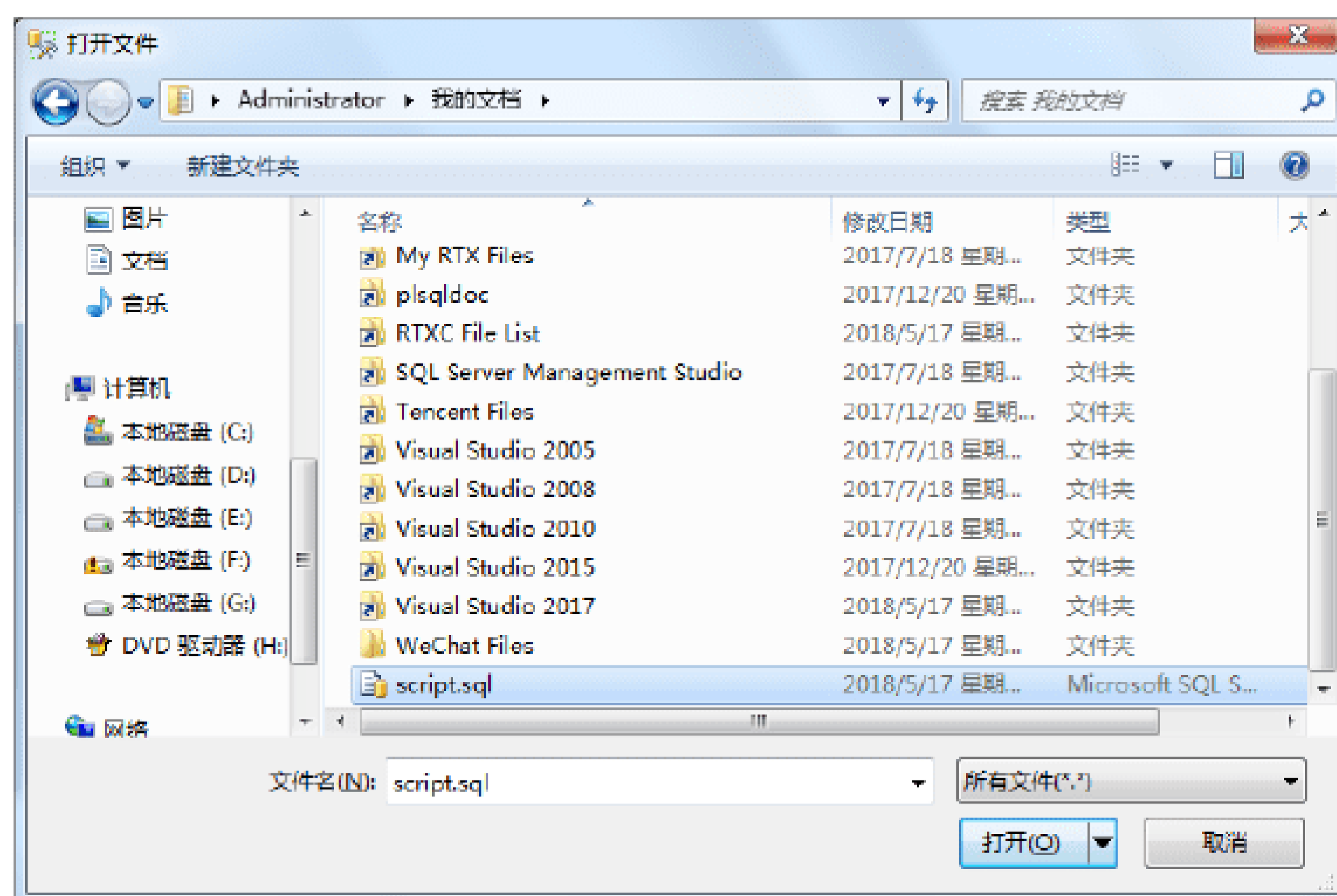


图 2.26　“打开文件”对话框

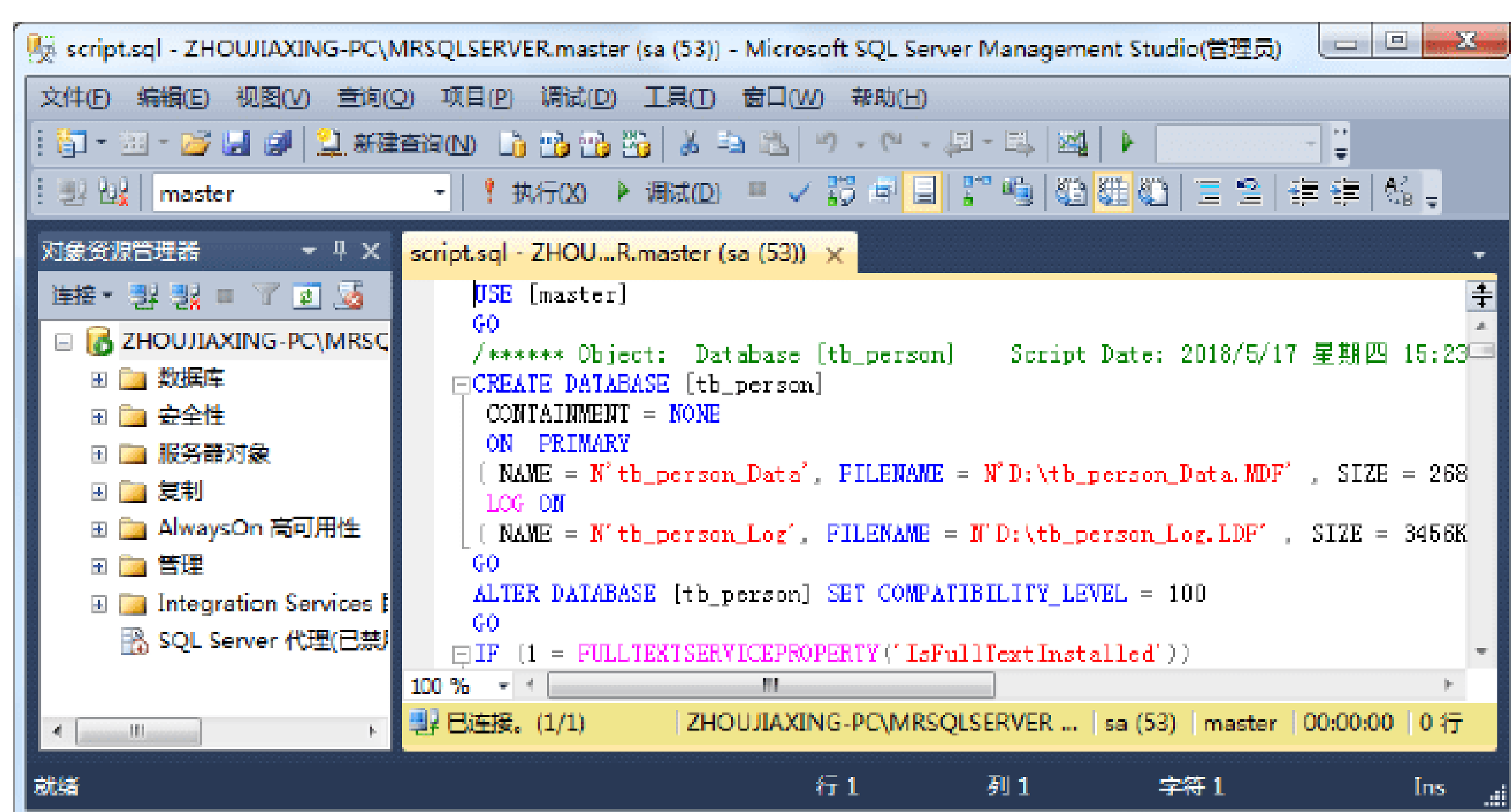


图 2.27　加载数据库脚本

2.4.4　批处理

批处理就是一个或多个 Transact-SQL 语句的集合，当要完成的任务不能由单独的 Transact-SQL 语句来完成时，可以使用批处理来组织多条 Transact-SQL 语句。从应用程序一次性发送到 SQL Server 并由 SQL Server 编译成一个可执行单元，此单元称为执行计划。执行计划中的语句每次只能执行一条。

建立批处理时，使用 GO 语句作为批处理的结束标记。但是在一个 GO 语句中只能使用注释文字不能包含其他 Transact-SQL 语句。如果在一个批处理中包含任何语法错误，例如，引用了一个并不存在的对象，则整个批处理就不能被成功地编译和执行。如果一个批处理中某句有执行错误，如违反了约束，它仅影响该句的执行，并不影响批处理中其他语句的执行。

建立批处理时，应当注意以下几点。

（1）CREATE DEFAULT、CREATE PROCEDURE、CREATE RULE、CREATE TRIGGER 及 CREATE VIEW 不能与其他语句放在一个批处理中。

（2）不能在一个批处理中引用其他批处理中所定义的变量。

（3）不能把规则和默认值绑定到表字段或用户自定义数据类型上之后，立即在同一个批处理中使用它们。

（4）不能定义一个 CHECK 约束之后，立即在同一个批处理中使用该约束。

（5）不能在修改表中的一个字段名之后，立即在同一个批处理中引用新字段名。

（6）如果一个批处理中的第一个语句是执行某个存储过程的 EXECUTE 语句，则 EXECUTE 关键字可以省略；如果该语句不是第一个语句，则必须使用 EXECUTE 关键字，或省略写为 EXEC。

2.5 备份和还原数据库

2.5.1 备份和恢复的概念

备份数据库是指对数据库或事务日志进行复制，当系统、磁盘或数据库文件损坏时，可以使用备份文件进行恢复，防止数据丢失。

还原数据库是使用数据库的备份文件对数据库进行还原操作。由于病毒的破坏、磁盘损坏或操作员操作失误等原因会导致数据丢失、不完整或数据错误，此时，需要对数据库进行还原，将数据还原到某一天，前提是当天必须进行了数据备份。

2.5.2 数据库备份

（1）打开 SQL Server Management Studio，在“对象资源管理器”中展开“数据库”节点，选中欲备份的数据库，如图 2.28 所示。

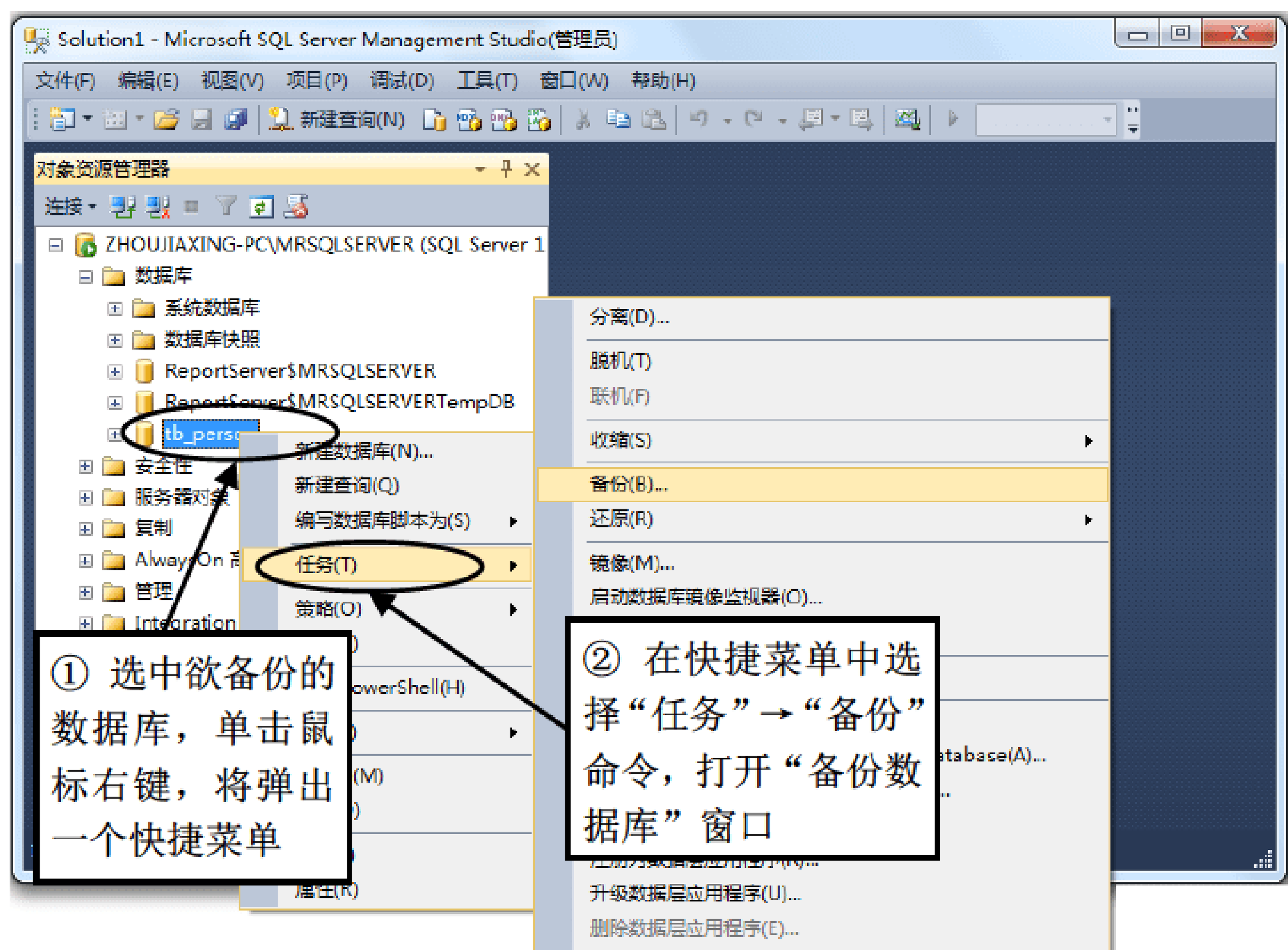


图 2.28 打开“备份数据库”窗口

（2）打开“备份数据库”窗口，为数据库指定备份文件，可以通过“添加”按钮更改备份文件的名称和磁盘位置，单击“确定”按钮开始备份数据库，如图 2.29 所示。

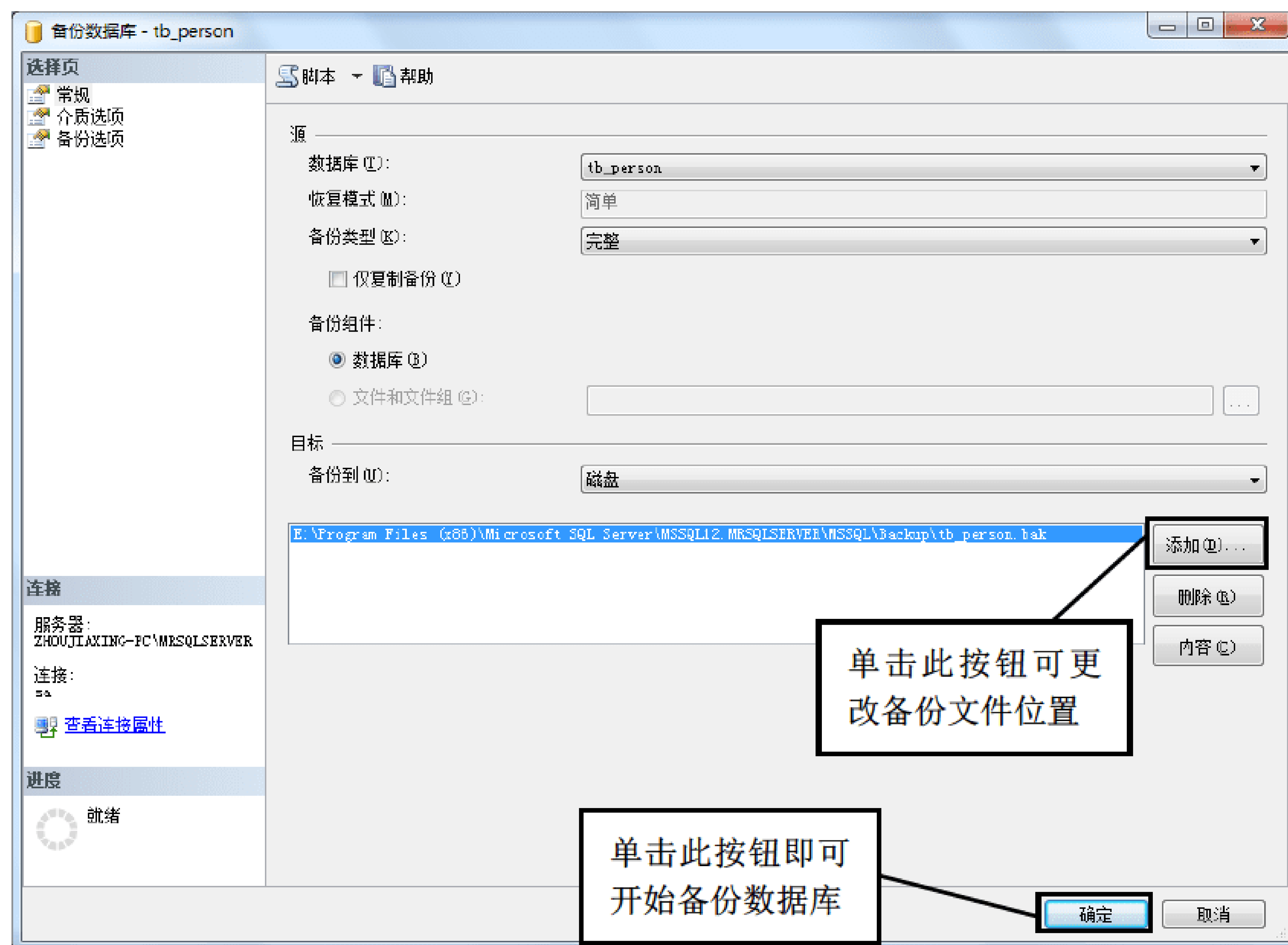


图 2.29　“备份数据库”窗口

2.5.3　数据库还原

打开 SQL Server Management Studio，在“对象资源管理器”中展开“数据库”节点，选中欲还原的数据库，如图 2.30 所示。

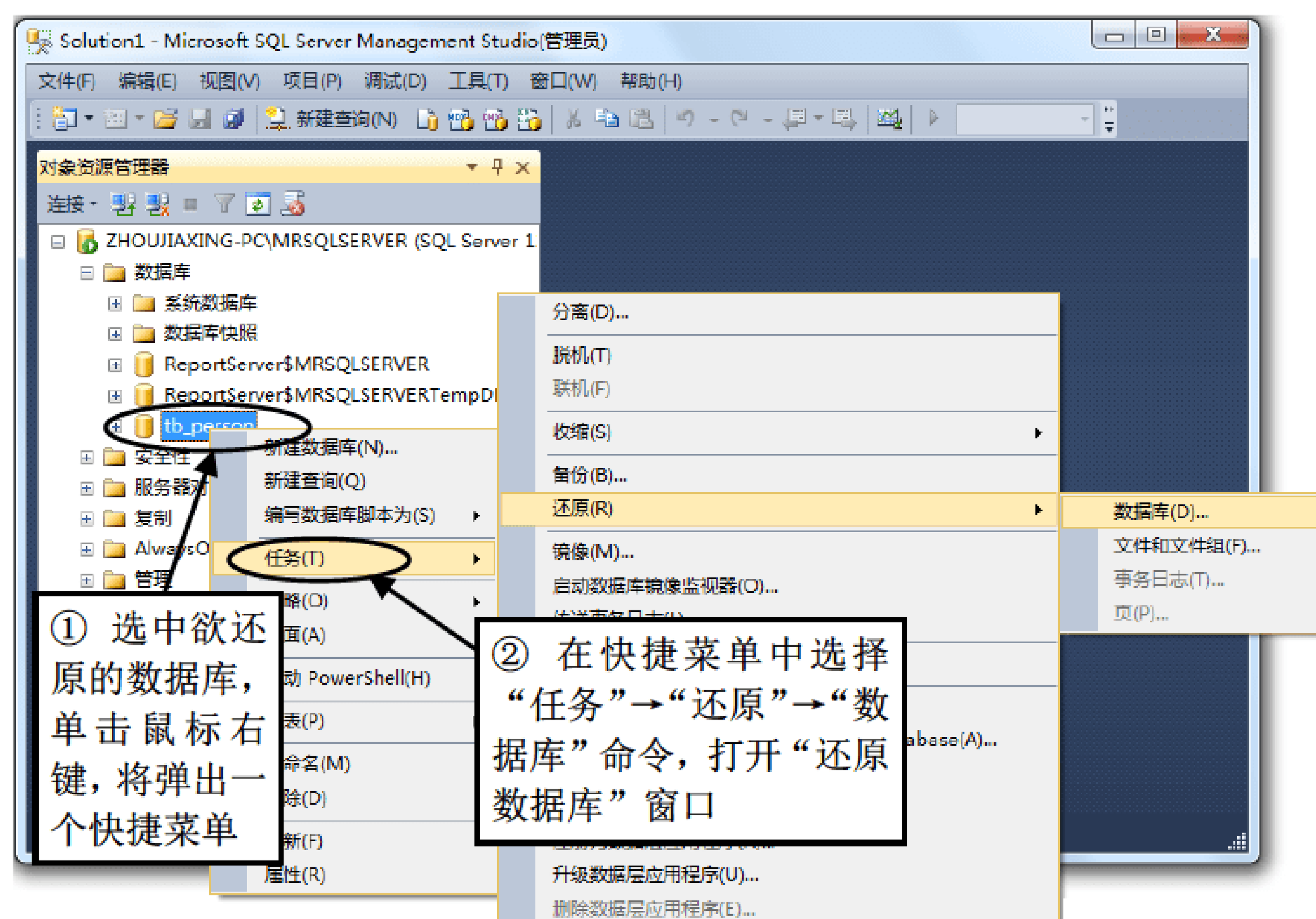


图 2.30　打开“还原数据库”窗口

2.6　分离和附加数据库

2.6.1　分离数据库

分离数据库是将数据库从服务器中分离出去，但并没有删除数据库，数据库文件依然存在，如果在需要使用数据库时，可以通过附加的方式将数据库附加到服务器中。在 SQL Server 2014 中分离数据库非常简单，方法如下：打开 SQL Server Management Studio，在“对象资源管理器”中展开“数据库”节点，选中欲分离的数据库，如图 2.31 所示。

图 2.31　打开“分离数据库”窗口

2.6.2　附加数据库

通过附加方式可以向服务器中添加数据库，前提是需要存在数据库文件和数据库日志文件。下面以附加 2.6.1 小节中分离的数据库为例介绍如何附加数据库。

打开 SQL Server Management Studio，在“对象资源管理器”中鼠标右键单击“数据库”节点，将弹出一个快捷菜单，按照如图 2.32 所示进行操作。

在打开的窗口中，单击“添加”按钮，选择要附加的数据库文件，依次单击“确定”按钮即可，如图 2.33 所示。

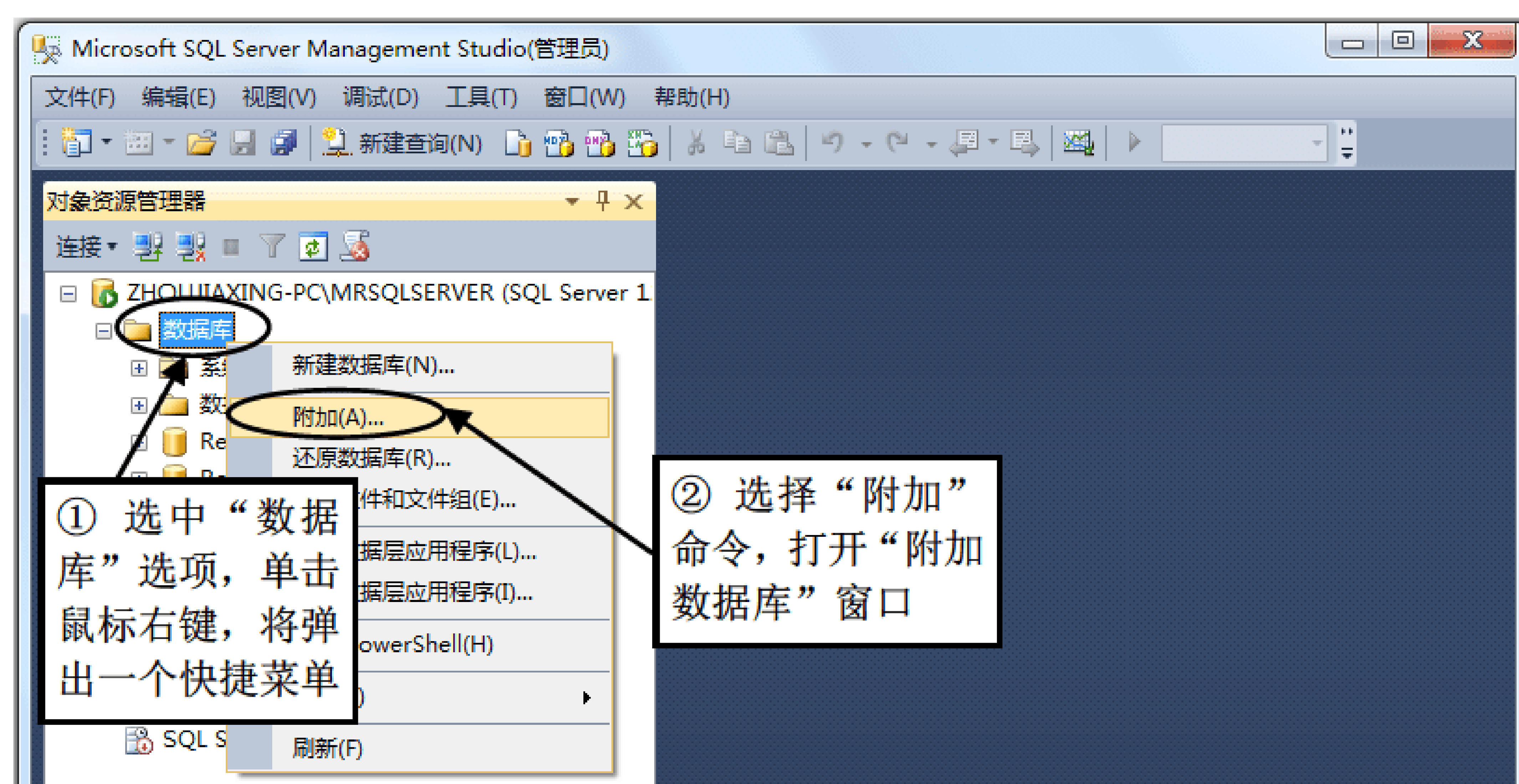


图 2.32　打开“附加数据库”窗口

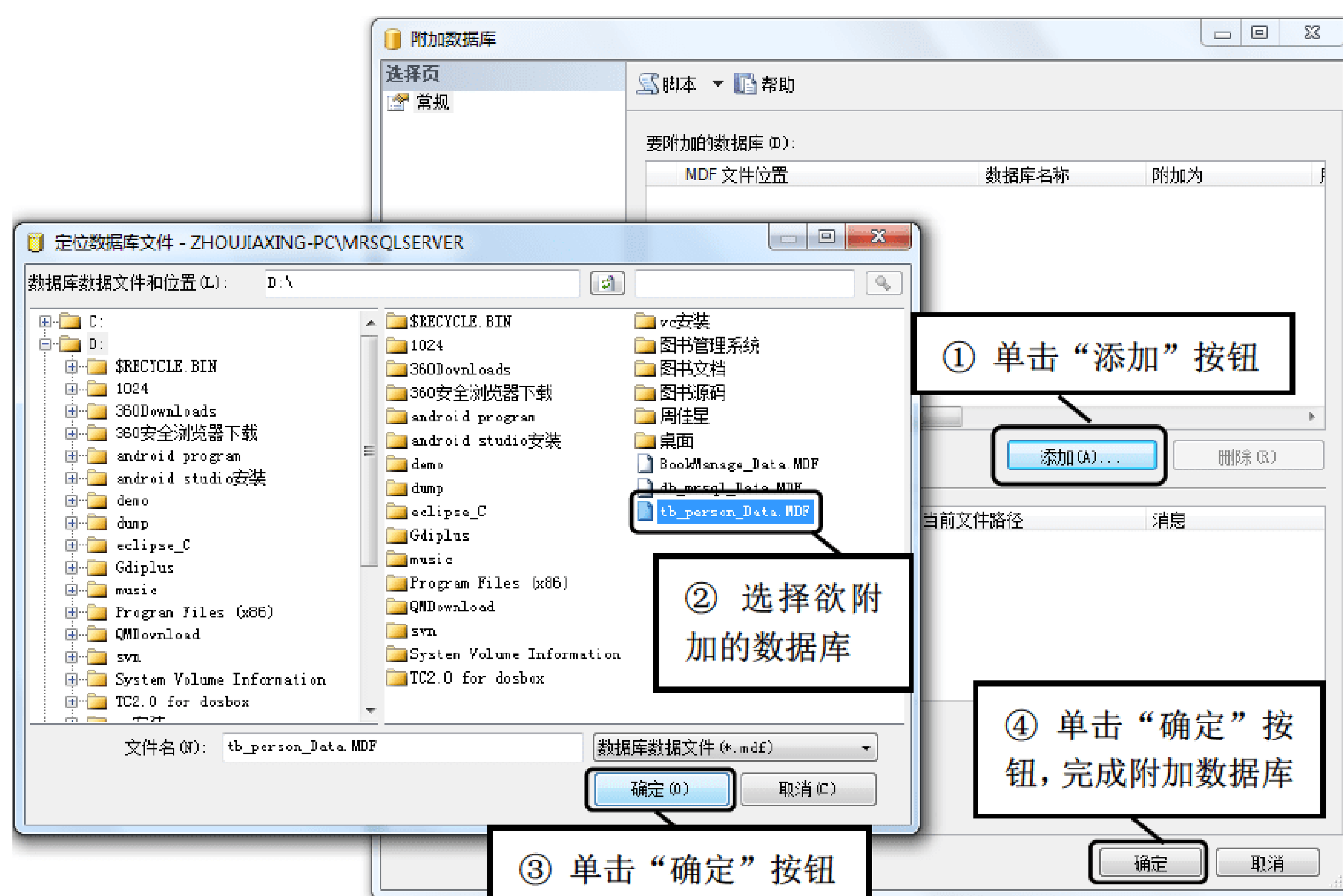


图 2.33　“附加数据库”窗口

2.7　导入和导出数据库或数据表

2.7.1　导入数据库

在 SQL Server 2014 中，用户可以将其他服务器中的数据库或数据表导入自己的系统中，而且在导入过程中可以选择自己需要的数据表，将其导入系统中，而不必将所有的数据表都导入系统中。

2.7.2　导入 SQL Server 数据表

在 SQL Server 2014 中导入数据非常方便，下面以向 tb_person 数据库中导入 BookManage 数据库中的 tb_bookinfo 表为例介绍如何导入数据表。

（1）打开 SQL Server Management Studio，在“对象资源管理器”中鼠标右键单击“数据库”节点，弹出如图 2.34 所示快捷菜单。

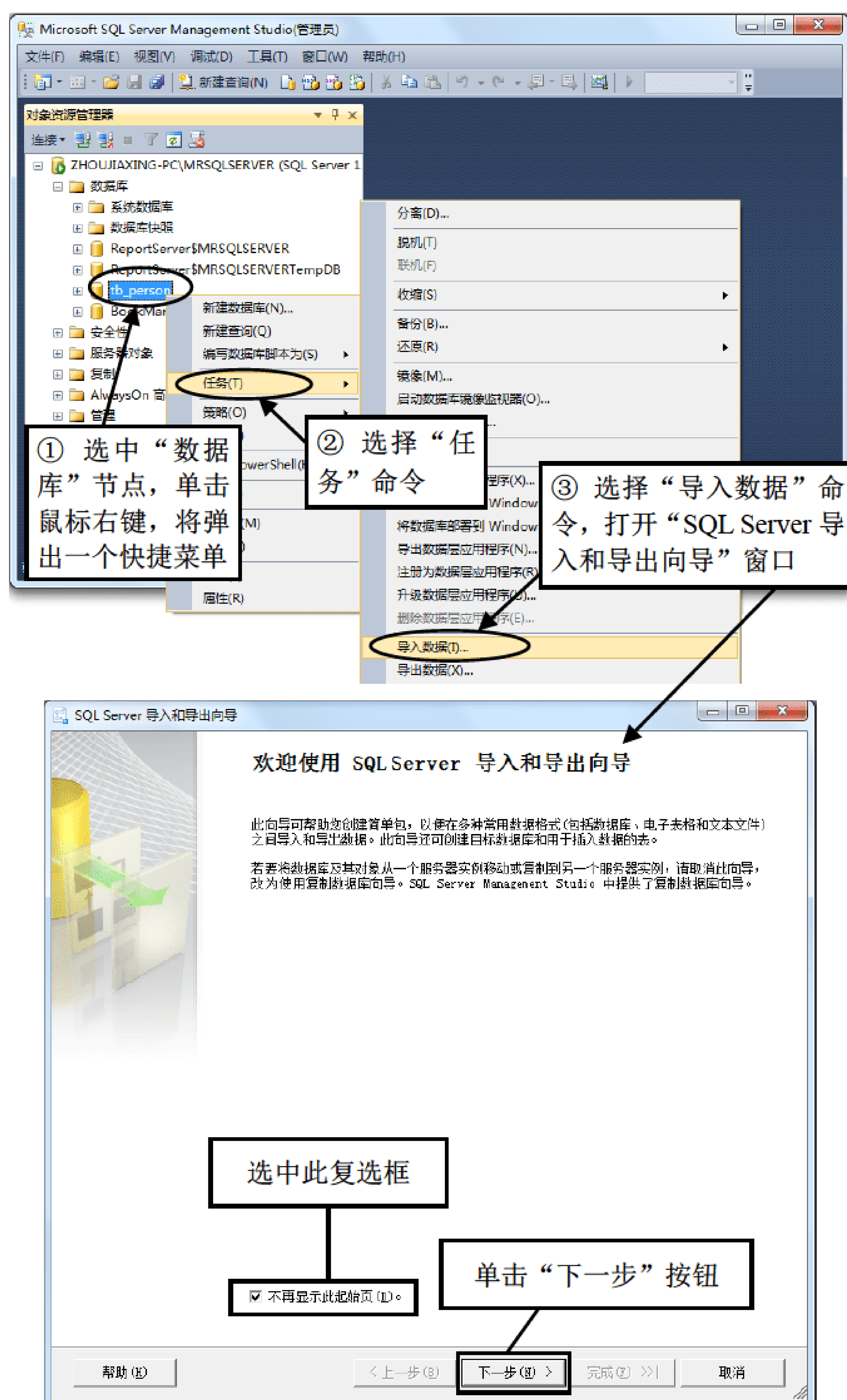


图 2.34　打开“SQL Server 导入和导出向导”窗口



或者选择“开始”→“所有程序”→Microsoft SQL Server 2014→SQL Server 2014 导入和导出数据，也能进入“SQL Server 导入和导出向导”窗口。

（2）选择数据源（本例为 BookManage 数据库）和目标数据库（本例为 tb_person 数据库），如图 2.35 和图 2.36 所示。

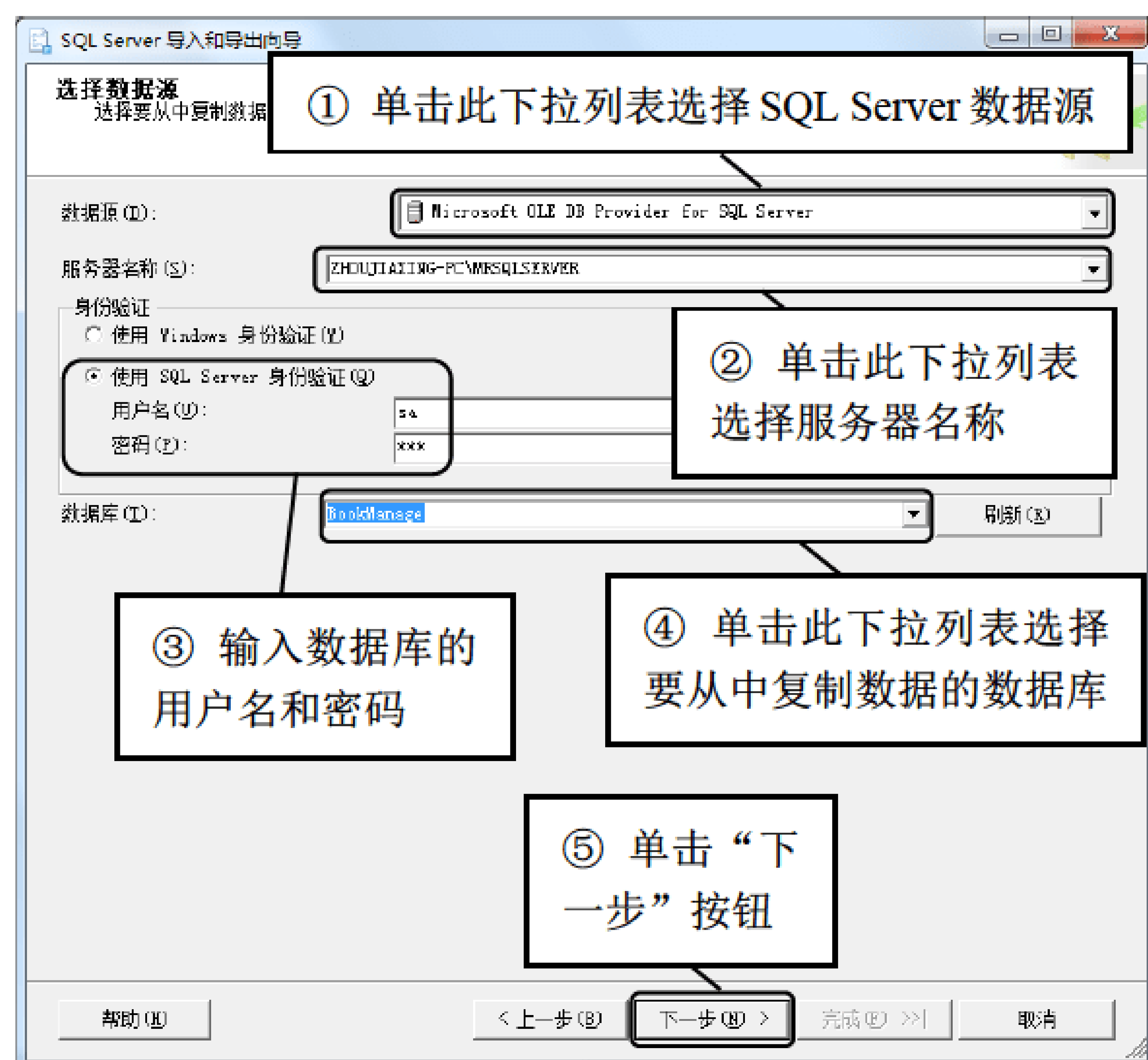


图 2.35　设置数据源

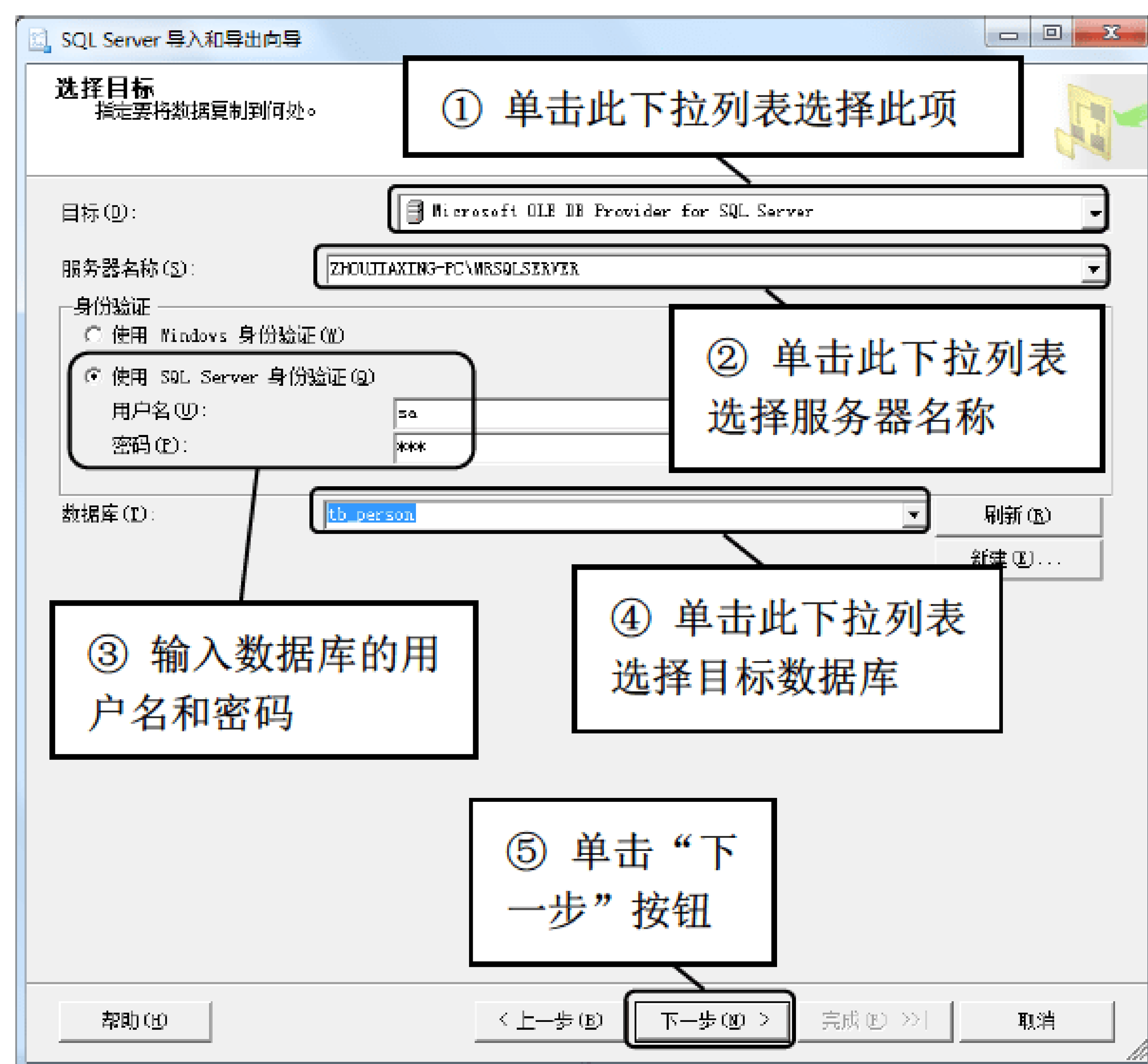


图 2.36　设置目标数据库

（3）选中“复制一个或多个表或视图的数据”单选按钮，单击“下一步”按钮，如图 2.37 所示，进入如图 2.38 所示的窗口，选择要复制的数据表，然后单击“下一步”按钮。

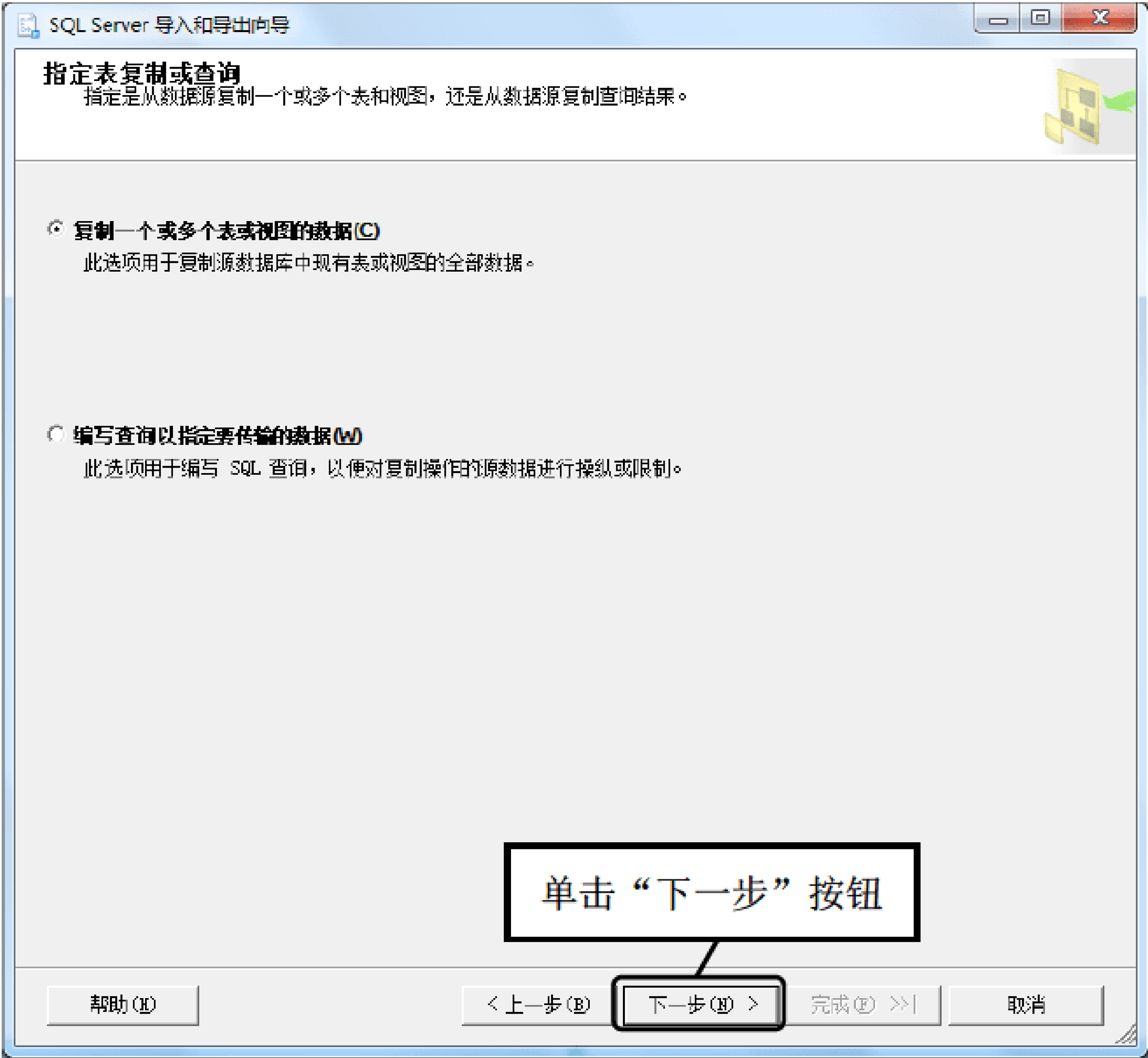


图 2.37　复制一个或多个表或视图的数据

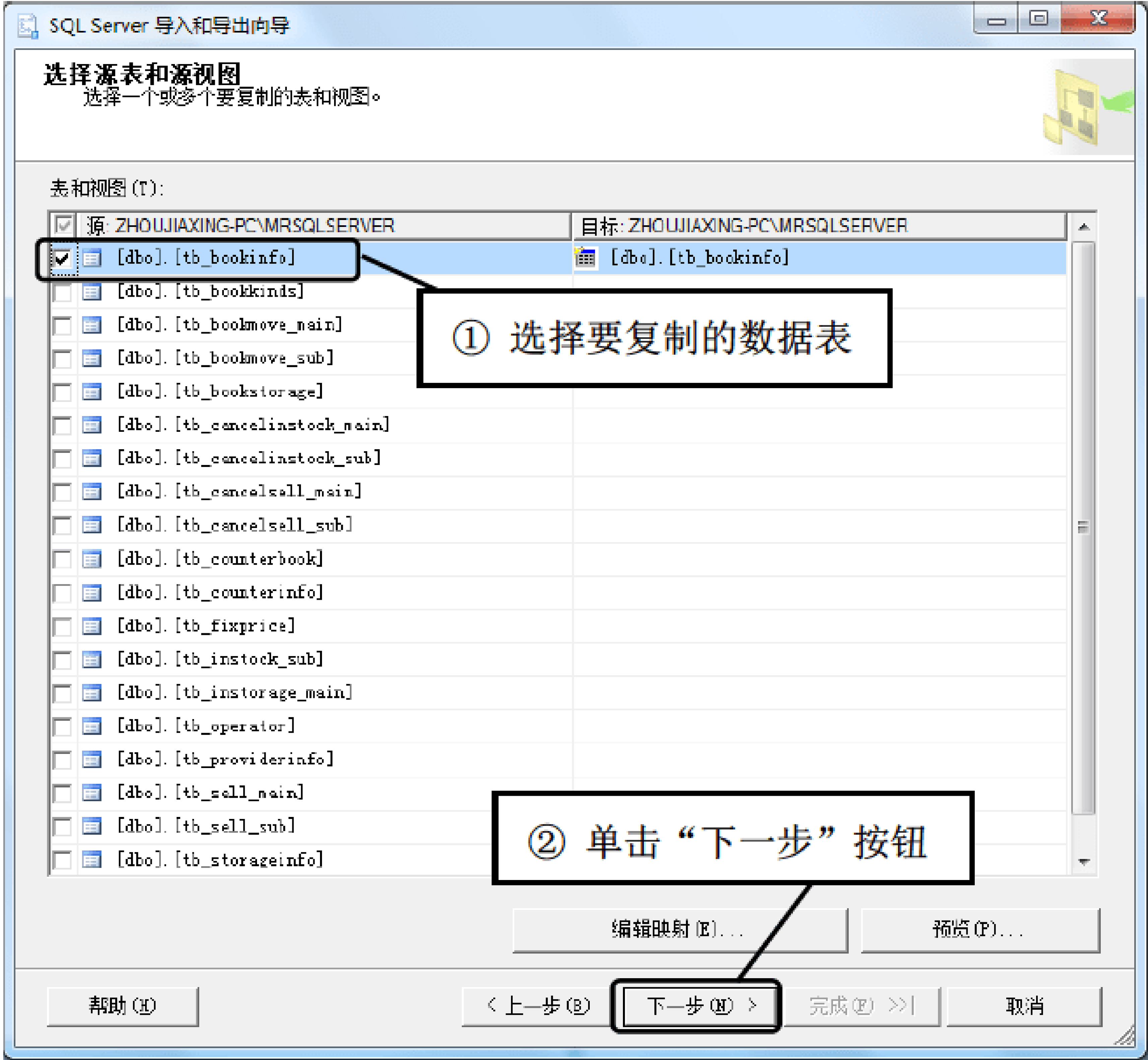


图 2.38　选择要复制的数据表

（4）在“保存并运行包”界面中单击“完成”按钮，开始复制数据，实现数据表的导入。

2.7.3　导入其他数据源的数据表

使用 SQL Server 导入/导出工具还能够将其他数据库中的数据表导入 SQL Server 中。

（1）实现导入其他数据源的数据表的步骤，与 2.7.2 小节中导入 SQL Server 数据表的步骤大致类似，只有步骤（2）选择数据源不一致，在设置数据源时，选择一个 Access 数据库作为数据源，如图 2.39 所示。

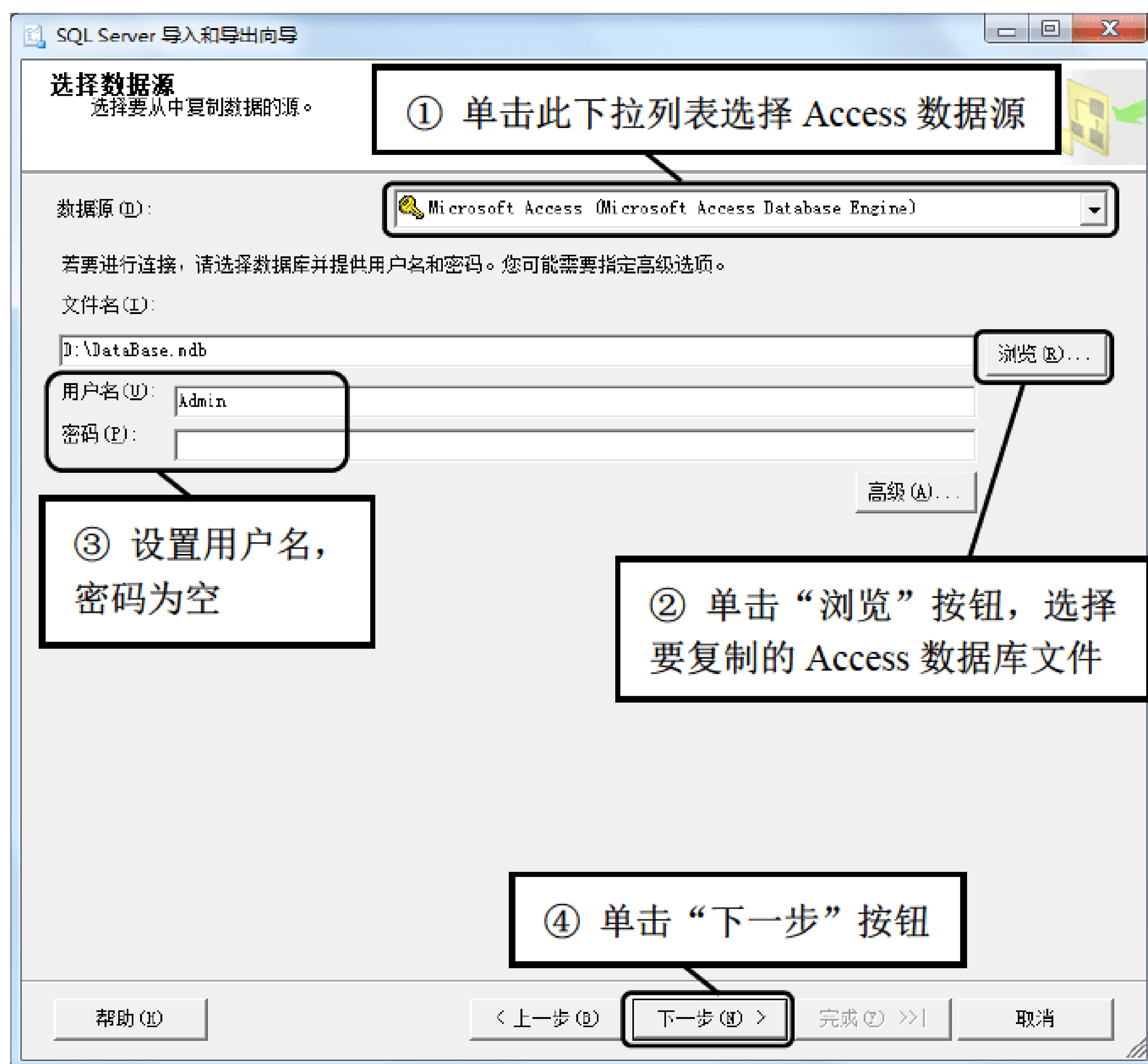


图 2.39　设置数据源

（2）在设置好数据源和目标数据库之后，需要选择要导入的数据表，如图 2.40 所示，要导入的数据表为 employees，单击“完成”按钮完成数据表的导入。

2.7.4　导出数据库

在 SQL Server 2014 中可以实现将本地服务器或远程服务器中的数据导出到另一个服务器中，它与导入数据是相对的。

2.7.5　导出 SQL Server 数据表

下面的例子是将 tb_person 数据库中的 tab_Employees 表导出到 BookManage 数据库中，具体步骤

如下。

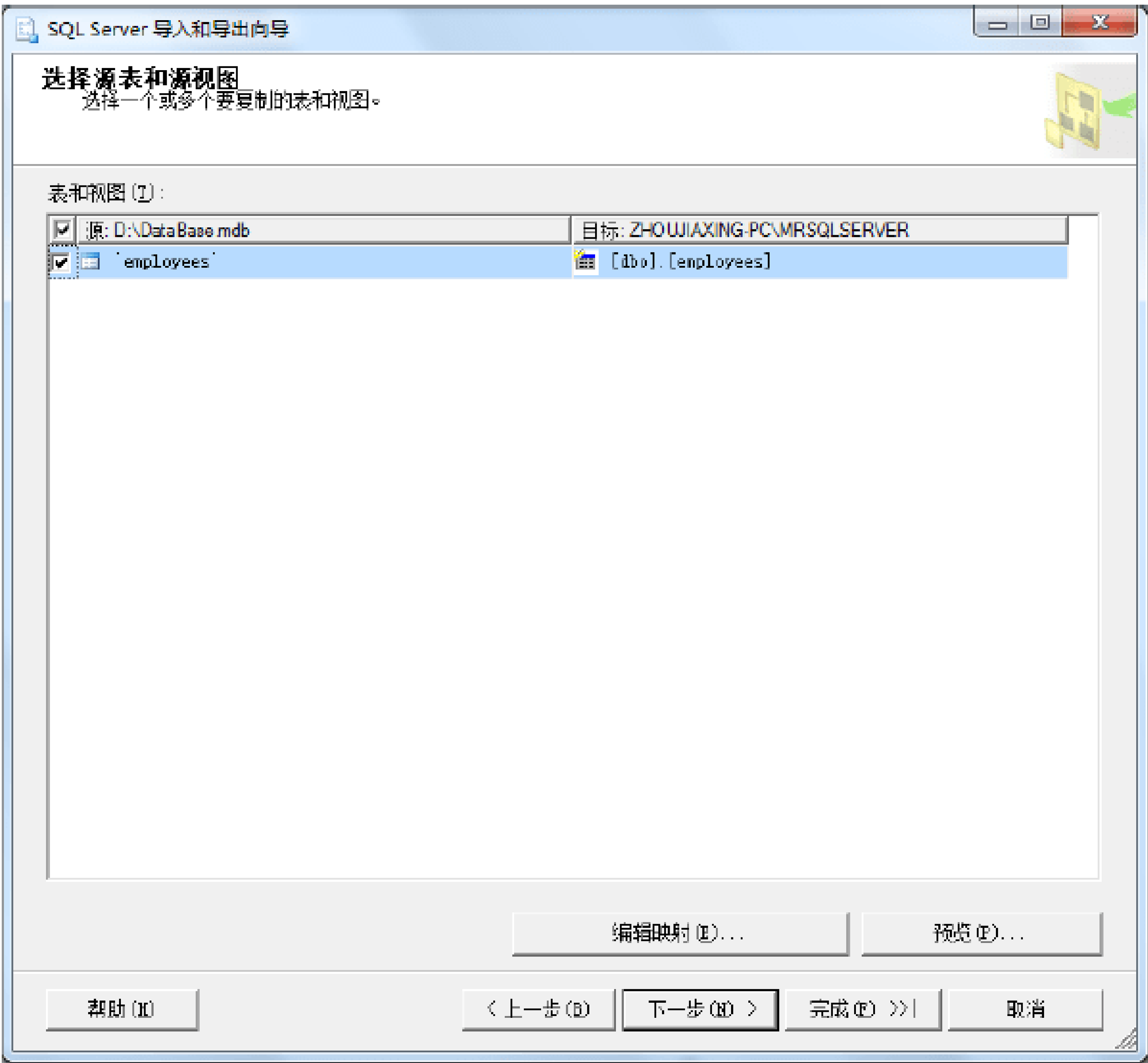


图 2.40　人事管理系统数据表

（1）选择“开始”→“所有程序”→Microsoft SQL Server 2014→SQL Server 2014 导入和导出数据，进入“SQL Server 导入和导出向导”窗口，设置数据源与目标数据库，如图 2.41 和图 2.42 所示。

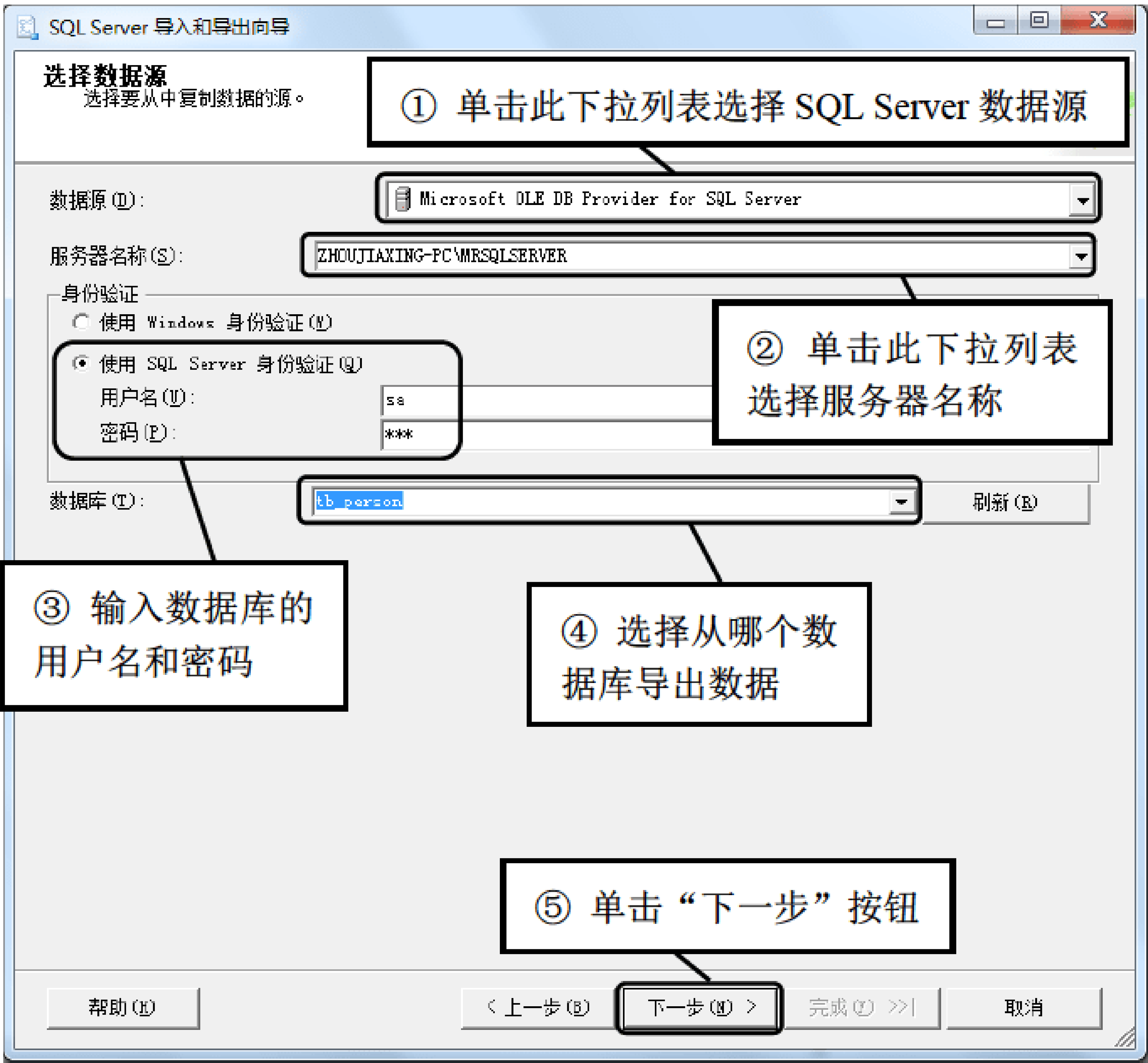


图 2.41　设置数据源

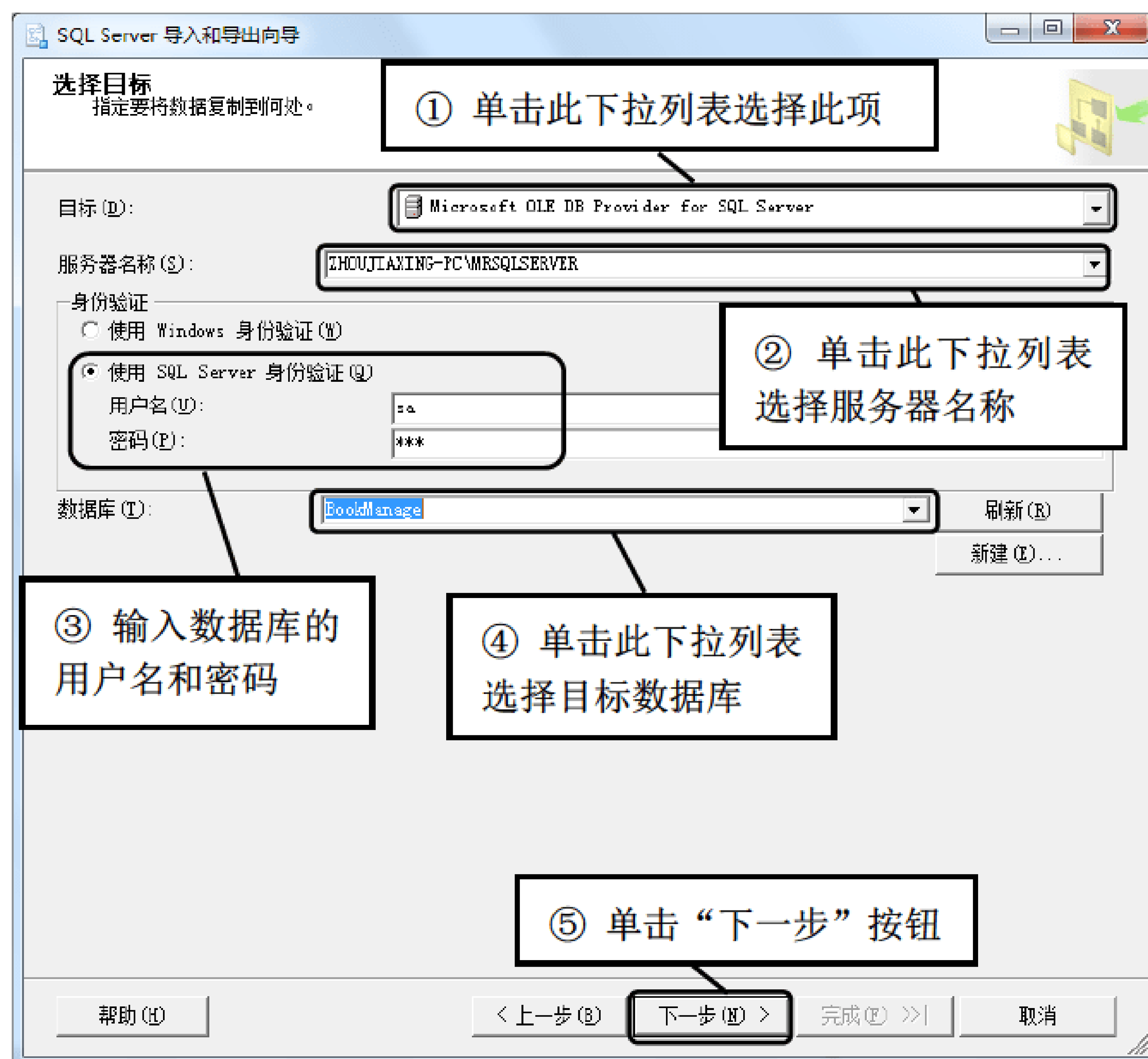


图 2.42　设置目标数据库

（2）选中“复制一个或多个表或视图的数据”单选按钮，单击“下一步”按钮，如图 2.43 所示，进入如图 2.44 所示的窗口，选择要复制的数据表，然后单击“下一步”按钮。

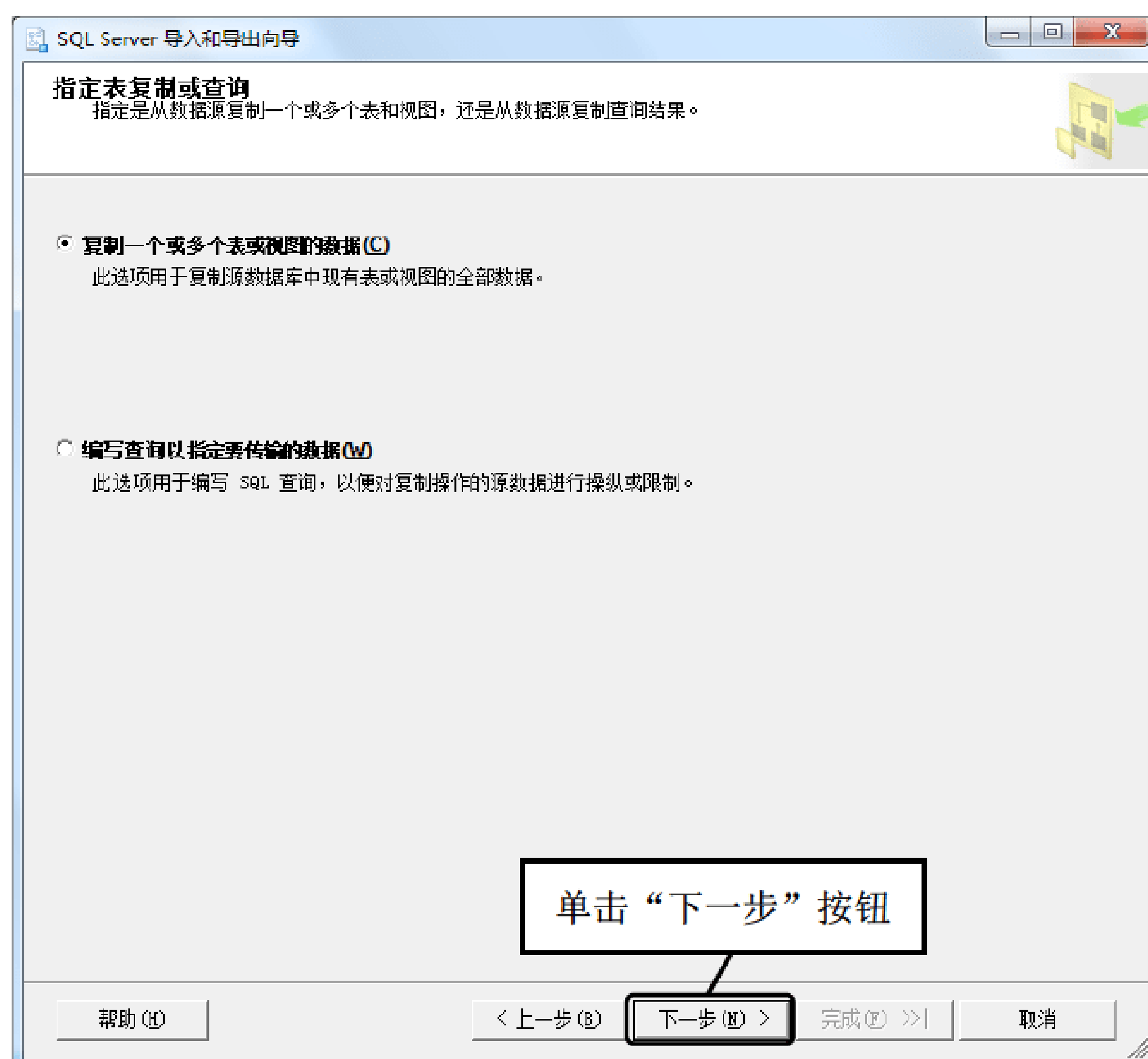


图 2.43　导出数据表的数据

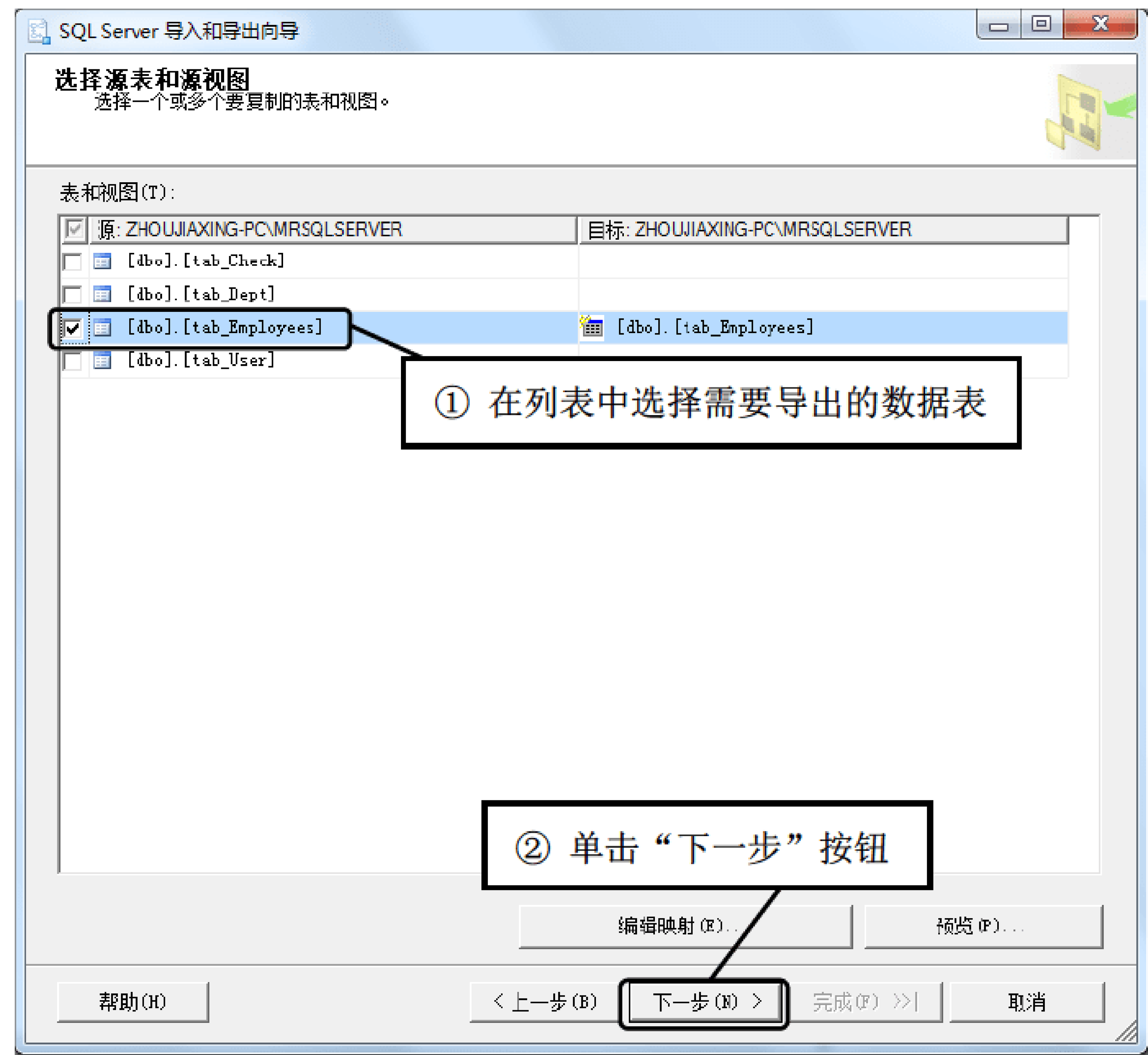


图 2.44　选择要导出的数据表

（3）在“保存并运行包”界面中单击“完成”按钮，开始导出数据。

2.8　小　　结

本章主要介绍 SQL Server 2014 的概念、安装与配置。在本地计算机上选择合适的版本安装 SQL Server 2014，以便能更好地配置 SQL Server 2014 连接服务器。配置成功后，还需要了解如何备份和还原数据库、分离和附加数据库、导入和导出数据库或数据表。

第 3 章

创建和管理数据库

（视频讲解：24 分钟）

本章主要介绍使用Transact-SQL语句和使用SQL Server Management Studio创建数据库、修改数据库和删除数据库的过程。通过本章的学习，读者可以熟悉SQL Server 2014数据库的组成元素，并能够掌握创建和管理数据库的方法，本章将详细讲解创建、修改、删除数据库的知识。

学习摘要：

- **数据库的基础知识**
- **SQL Server 的命名规则**
- **创建、修改和删除数据库**


视频讲解

3.1 认识数据库

Microsoft SQL Server 2014 数据库同 Microsoft 的其他数据库类似，主要应用存储数据及其相同的对象（如视图、索引、存储过程和触发器等），以便随时对数据库中的数据及其对象进行访问和管理。本节将对数据库的基本概念、数据库对象及其相关知识进行详细的介绍。

3.1.1 数据库基本概念

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库，是存储在一起的相关数据的集合。其优点主要体现在以下几方面。

（1）减少数据的冗余度，节省数据的存储空间。

（2）具有较高的数据独立性和易扩充性。

（3）实现数据资源的充分共享。

下面介绍一下与数据库相关的几个概念。

（1）数据库系统

数据库系统（Database System，DBS）是采用数据库技术的计算机系统，是由数据库（数据）、数据库管理系统（软件）、数据库管理员（人员）、硬件平台（硬件）和软件平台（软件）5 部分构成的运行实体。其中，数据库管理员（Database Administrator，DBA）是对数据库进行规划、设计、维护和监视等操作的专业管理人员，在数据库系统中起着非常重要的作用。

（2）数据库管理系统

数据库管理系统（Database Management System，DBMS）是数据库系统的一个重要组成部分，是位于用户与操作之间的一个数据管理软件，负责数据库中的数据组织、数据操纵、数据维护和数据服务等。主要具有如下功能。

① 数据存取的物理构建：为数据模式的物理存取与构建提供有效的存取方法与手段。

② 数据操纵功能：为用户使用数据库的数据提供方便，如查询、插入、修改、删除以及简单的算术运算和统计。

③ 数据定义功能：用户可以通过数据库管理系统提供的数据定义语言（Data Definition Language，DDL）方便地对数据库中的对象进行定义。

④ 数据库的运行管理：数据库管理系统统一管理数据库的运行和维护，以保障数据的安全性、完整性、并发性和故障的系统恢复性。

⑤ 数据库的建立和维护功能：数据库管理系统能够完成初始数据的输入和转换、数据库的转储和恢复、数据库的性能监视和分析等任务。

（3）关系数据库

关系数据库是支持关系模型的数据库。关系模型由关系数据结构、关系操作集合和完整性约束 3 部分组成。

① 关系数据结构：在关系模型中数据结构单一，现实世界的实体以及实体间的联系均用关系来表示，实际上关系模型中的数据结构就是一张二维表。

② 关系操作集合：关系操作分为关系代数、关系演算、具有关系代数和关系演算双重特点的语言（SQL）。

③ 完整性约束：完整性约束包括实体完整性、参照完整性和用户定义的完整性。

3.1.2　数据库常用对象

在 SQL Server 2014 的数据库中，表、索引、视图和存储过程等具体存储数据或对数据进行操作的实体都被称为数据库对象。下面介绍几种常用的数据库对象。

（1）表

表是包含数据库中所有数据的数据库对象，由行和列组成，用于组织和存储数据。

（2）字段

表中每列称为一个字段，字段具有自己的属性，如字段类型、字段大小等，其中字段类型是字段最重要的属性，它决定了字段能够存储哪种数据。

SQL 规范支持 5 种基本字段类型：字符型、文本型、数值型、逻辑型和日期时间型。

（3）索引

索引是一个单独的、物理的数据库结构。它是依赖于表建立的，在数据库中索引使数据库程序无须对整个表进行扫描，就可以在其中找到所需的数据。

（4）视图

视图是从一张或多张表中导出的表（也称虚拟表），是用户查看数据表中数据的一种方式。表中包括几个被定义的数据列与数据行，其结构和数据建立在对表的查询基础之上。

（5）存储过程

存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集合（包含查询、插入、删除和更新等操作），经编译后以名称的形式存储在 SQL Server 服务器端的数据库中，由用户通过指定存储过程的名字来执行。当这个存储过程被调用执行时，这些操作也会同时执行。

3.1.3　数据库组成

SQL Server 2014 数据库主要由文件和文件组组成。数据库中的所有数据和对象（如表、存储过程和触发器）都被存储在文件中。

（1）文件

文件主要分为以下 3 种类型。

① 主要数据文件：存放数据和数据库的初始化信息。每个数据库有且只有一个主要数据文件，默认扩展名是.mdf。

② 次要数据文件：存放除主要数据文件以外的所有数据文件。有些数据库可能没有次要数据文件，也可能有多个次要数据文件，默认扩展名是.ndf。

③ 事务日志文件：存放用于恢复数据库的所有日志信息。每个数据库至少有一个事务日志文件，

也可以有多个事务日志文件，默认扩展名是.ldf。

（2）文件组

文件组是 SQL Server 2014 数据文件的一种逻辑管理单位，它将数据库文件分成不同的文件组，方便于对文件的分配和管理。

文件组主要分为以下两种类型。

① 主文件组：包含主要数据文件和任何没有明确指派给其他文件组的文件。系统表的所有页都分配在主文件组中。

② 用户定义文件组：主要是在 CREATE DATABASE 或 ALTER DATABASE 语句中，使用 FILEGROUP 关键字指定的文件组。


说明

每个数据库中都有一个文件组作为默认文件组运行，默认文件组包含在创建时没有指定文件组的所有表和索引的页。在没有指定的情况下，主文件组作为默认文件组。

对文件进行分组时，一定要遵循文件和文件组的设计规则。

① 文件只能是一个文件组的成员。

② 文件或文件组不能由一个以上的数据库使用。

③ 数据和事务日志信息不能属于同一文件或文件组。

④ 日志文件不能作为文件组的一部分。日志空间与数据空间分开管理。


注意

系统管理员在进行备份操作时，可以备份或恢复个别的文件或文件组，而不用备份或恢复整个数据库。

3.1.4　系统数据库

SQL Server 2014 的安装程序在安装时默认将建立 4 个系统数据库（master、tempdb、model、msdb）。下面分别对其进行介绍。

（1）master 数据库

SQL Server 2014 中最重要的数据库。记录 SQL Server 实例的所有系统级信息，包括实例范围的元数据、端点、链接服务器和系统配置设置。

（2）tempdb 数据库

tempdb 是一个临时数据库，用于保存临时对象或中间结果集。

（3）model 数据库

用作 SQL Server 实例上创建的所有数据库的模板。对 model 数据库进行的修改（如数据库大小、排序规则、恢复模式和其他数据库选项）将应用于以后创建的所有数据库。

（4）msdb 数据库

用于 SQL Server 代理计划警报和作业。

3.2 SQL Server 的命名规范



SQL Server 为了完善数据库的管理机制，设计了严格的命名规则。用户在创建数据库及数据库对象时必须严格遵守 SQL Server 的命名规则。本节将对标识符、对象和实例的命名进行详细的介绍。

3.2.1 标识符

在 SQL Server 中，服务器、数据库和数据库对象（如表、视图、列、索引、触发器、过程、约束和规则等）都有标识符，数据库对象的名称被看成是该对象的标识符。大多数对象要求带有标识符，但有些对象（如约束）中标识符是可选项。

对象标识符是在定义对象时创建的，标识符随后用于引用该对象，下面分别对标识符的格式及分类进行介绍。

1. 标识符格式

在定义标识符时必须遵守以下规定。

（1）标识符的首字符必须是下列字符之一。

☑ 统一码（Unicode）2.0 标准中所定义的字母，包括拉丁字母 a～z 和 A～Z，以及来自其他语言的字符。

☑ 下画线“_”、at 符号“@”或者数字符号“#”。

在 SQL Server 中，某些处于标识符开始位置的符号具有特殊意义。以 at 符号“@”开始的标识符表示局部变量或参数；以一个数字符号“#”开始的标识符表示临时表或过程，如表“#gzb”就是一张临时表；以双数字符号“##”开始的标识符表示全局临时对象，如表“##gzb”则是全局临时表。


注意

某些 Transact-SQL 函数的名称以双 at 符号（@@）开始，为避免混淆这些函数，建议不要使用以@@开始的名称。

（2）标识符的后续字符可以是以下 3 种。

☑ 统一码（Unicode）2.0 标准中所定义的字母。

☑ 来自拉丁字母或其他国家/地区脚本的十进制数字。

☑ at 符号“@”、美元符号“$”、数字符号“#”或下画线“_”。

（3）标识符不允许是 Transact-SQL 的保留字。

（4）不允许嵌入空格或其他特殊字符。

例如，为明日科技公司创建一个工资管理系统，可以将其数据库命名为 MR_GZGLXT。名字除了要遵守命名规则以外，最好还能准确表达数据库的内容，本例中的数据库名称是以每个字的大写字母命名的，其中还使用了下画线“_”。

2．标识符分类

SQL Server 将标识符分为以下两种类型。

☑ 常规标识符：符合标识符的格式规则。

☑ 分隔标识符：包含在双引号（" "）或者方括号（[ ]）内的标识符。该标识符可以不符合标识符的格式规则，如[MR GZGLXT]，MR 和 GZGLXT 之间含有空格，但因为使用了方括号，所以视为分隔标识符。


注意

常规标识符和分隔标识符包含的字符数必须在 1～128 之间，对于本地临时表，标识符最多可以有 116 个字符。

3.2.2 对象命名规则

SQL Server 2014 的数据库对象的名字由 1～128 个字符组成，不区分大小写。使用标识符也可以作为对象的名称。

在一个数据库中创建了一个数据库对象后，数据库对象的完整名称应该由服务器名、数据库名、拥有者名和对象名 4 部分组成，其格式如下：

```
[[[server.] [database] .] [owner_name] .] object_name
```

服务器、数据库和所有者的名称即所谓的对象名称限定符。当引用一个对象时，不需要指定服务器、数据库和所有者，可以利用句号标出它们的位置，从而省略限定符。

对象名的有效格式如下：

```
server.database.owner_name.object_name
server.database..object_name
server..owner_name.object_name
server...object_name
database.owner_name.object_name
database..object_name
owner_name.object_name
object_name
```

指定了 4 个部分的对象名称被称为完全合法名称。


注意

不允许存在 4 部分名称完全相同的数据库对象。在同一个数据库里可以存在两个名为 EXAMPLE 的表格，但前提必须是这两个表的拥有者不同。

3.2.3 实例命名规则

使用 SQL Server 2014，可以选择在一台计算机上安装 SQL Server 的多个实例。SQL Server 2014 提

供了两种类型的实例：默认实例和命名实例。

（1）默认实例

此实例由运行它的计算机的网络名称标识。使用以前版本 SQL Server 客户端软件的应用程序可以连接到默认实例。SQL Server 6.5 版或 SQL Server 7.0 版服务器可作为默认实例操作。但是，一台计算机上每次只能有一个版本作为默认实例运行。

（2）命名实例

计算机可以同时运行任意个 SQL Server 命名实例。实例通过计算机的网络名称加上实例名称以<计算机名称>\<实例名称>格式进行标识，即 computer_name\instance_name，但该实例名不能超过 16 个字符。

3.3　数据库操作



3.3.1　创建数据库

在 SQL Server 创建用户数据库之前，用户必须设计好数据库的名称以及它的所有者、空间大小和存储信息的文件和文件组。

1．以界面方式创建数据库

下面在 SQL Server Management Studio 中创建数据库 db_database，具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）鼠标右键单击“数据库”节点，在弹出的快捷菜单中选择“新建数据库”命令，如图 3.1 所示。

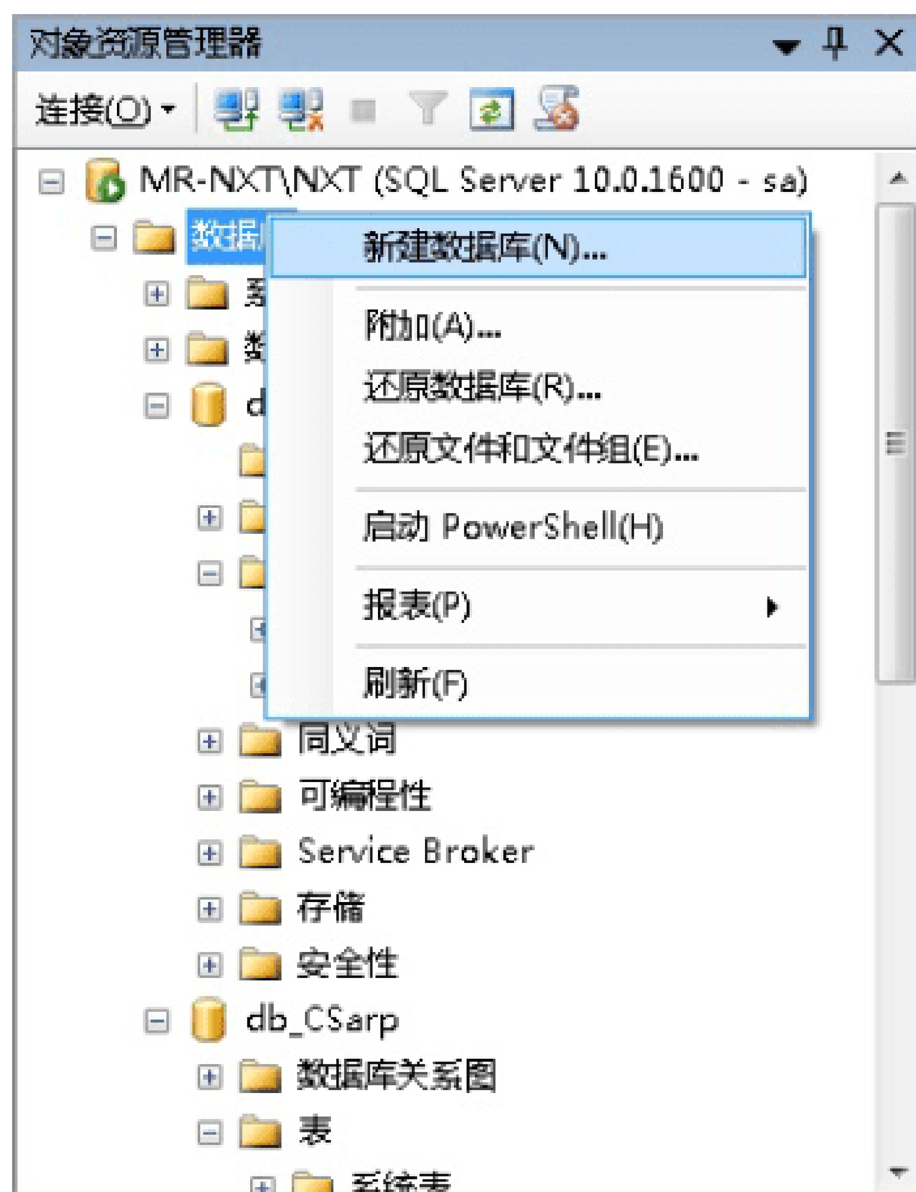


图 3.1　新建数据库

（3）进入“新建数据库”窗口，如图 3.2 所示。在列表框中填写数据库名 db_database，单击“确定”按钮，即添加数据库成功。

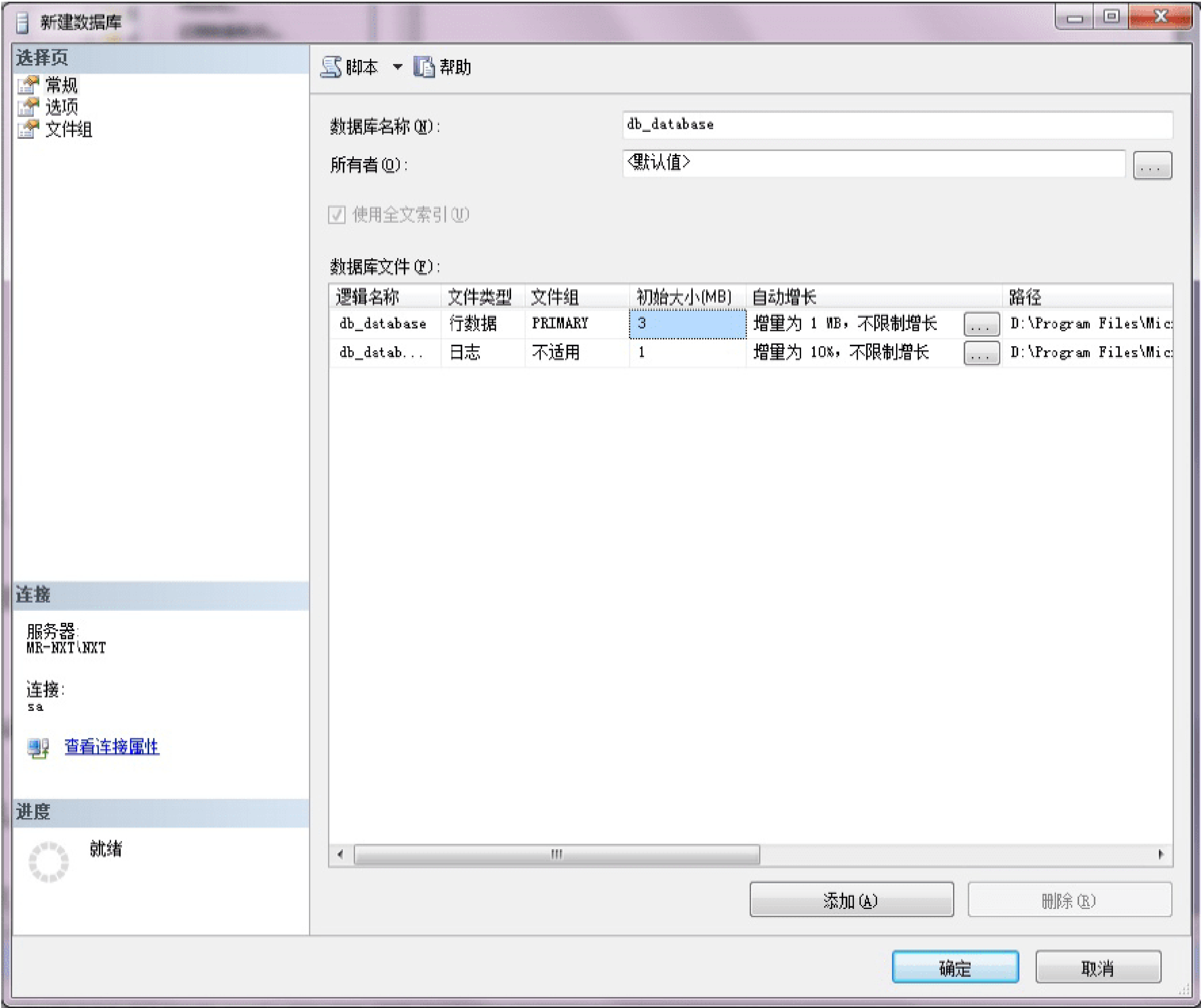


图 3.2　创建数据库名称

2．使用 CREATE DATABASE 语句创建数据库

语法格式如下：

```
CREATE DATABASE 数据库名
```

例如，使用命令创建超市管理系统数据库 db_supermarket。

```
create database db_supermarket      --使用 create database 命令创建一个名称为 db_supermarket 的数据库
```

运行的结果如图 3.3 所示。

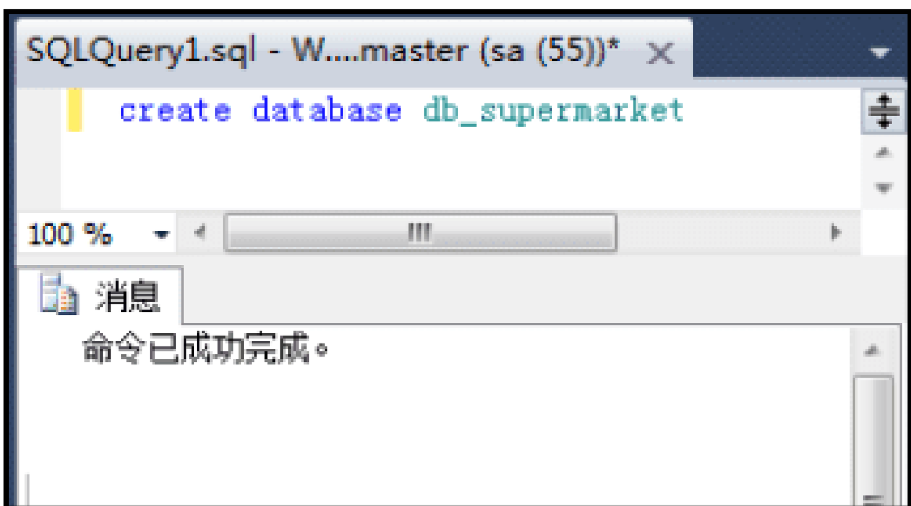


图 3.3　创建一个名称为 db_supermarket 的数据库



注意

在创建数据库时，所要创建的数据库名称必须是系统中不存在的，如果存在相同名称的数据库，在创建数据库时系统将会报错。另外，数据库的名称也可以是中文名称。

3.3.2　修改数据库

数据库创建完成后，常常需要根据用户环境进行调整，如对数据库的某些参数进行更改，这就需要使用修改数据库的命令。

1．以界面方式修改数据库

下面介绍如何更改数据库 db_2012 的所有者。具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库，在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击需要更改的数据库 db_2012 选项，在弹出的快捷菜单中选择“属性”命令，如图 3.4 所示。

（3）进入“数据库属性”窗口，如图 3.5 所示。通过该窗口可以修改数据库的相关选项。

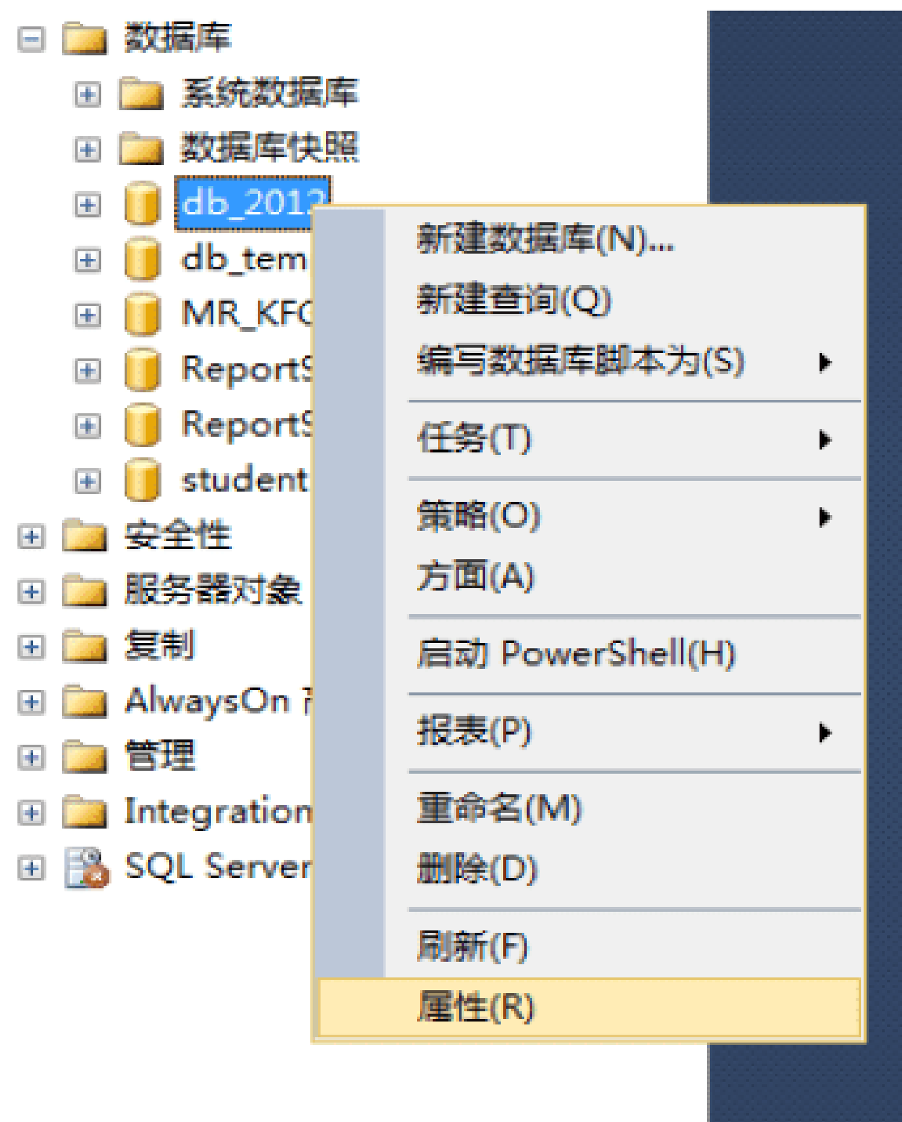


图 3.4　选择数据库属性

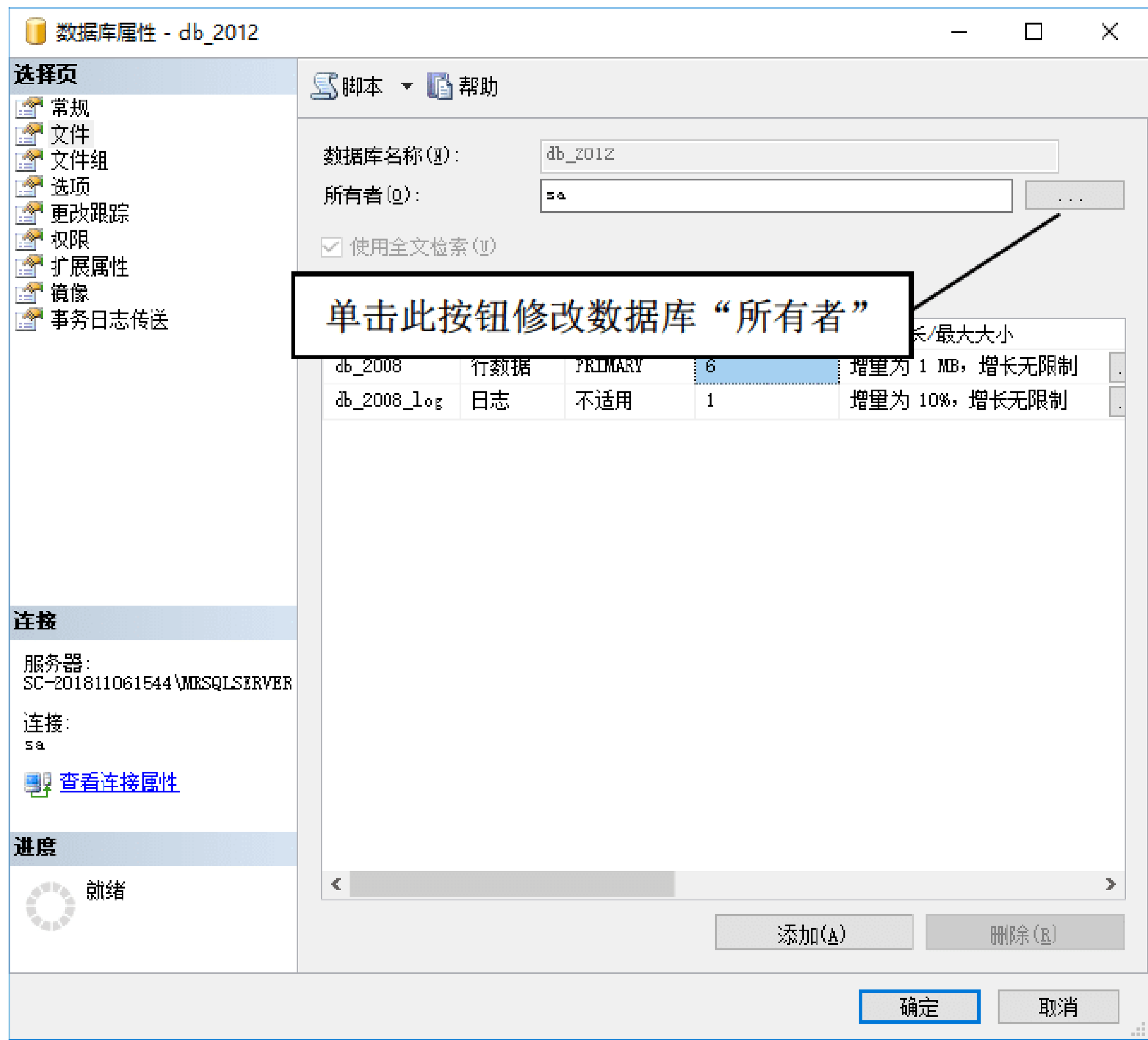


图 3.5　“数据库属性”窗口

（4）单击“数据库属性”窗口中的“文件”选项，然后单击“所有者”后的 ... 按钮，弹出“选择数据库所有者”对话框，如图 3.6 所示。

（5）单击“浏览”按钮，弹出“查找对象”对话框，如图 3.7 所示。通过该对话框选择匹配对象。

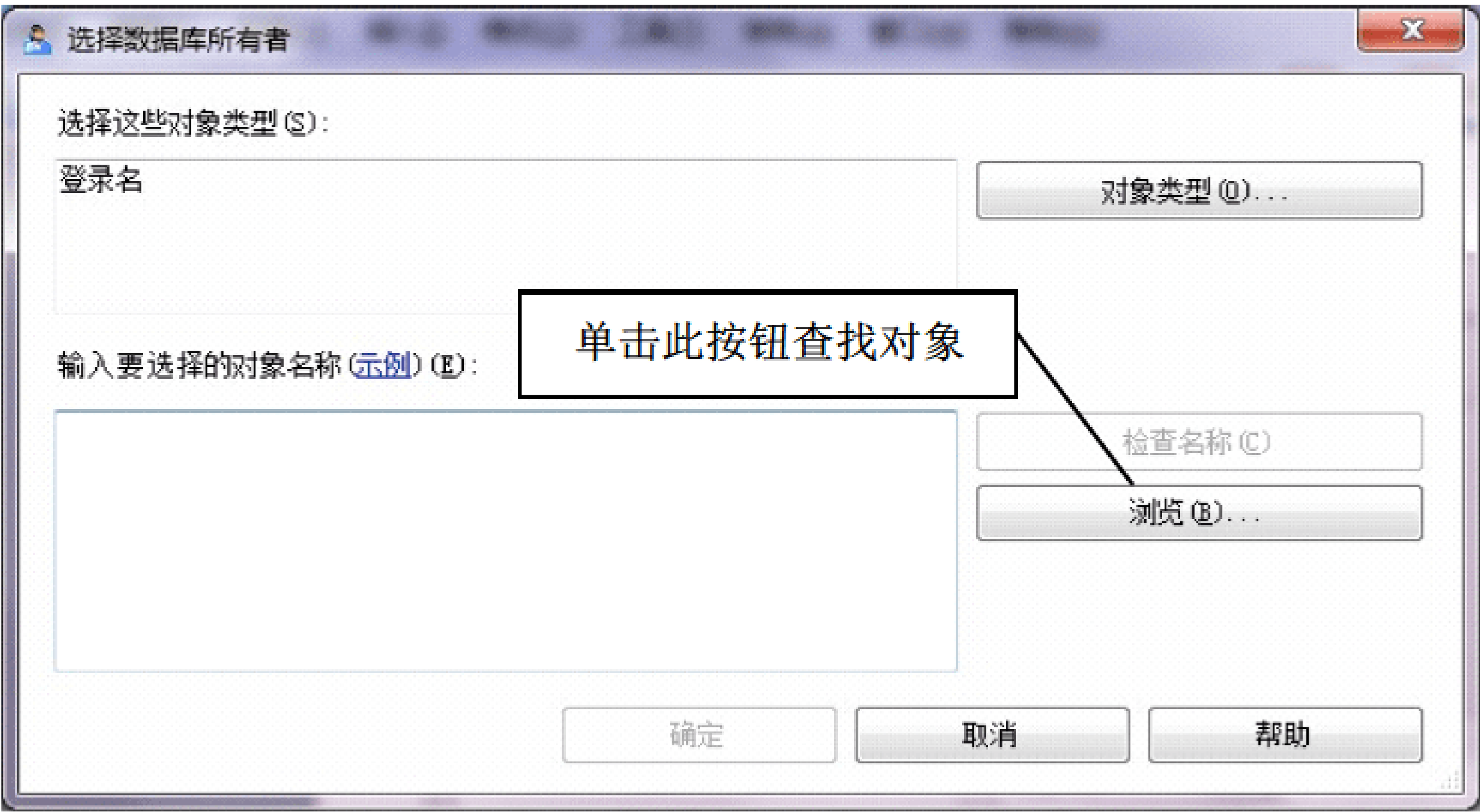


图 3.6　选择数据库所有者

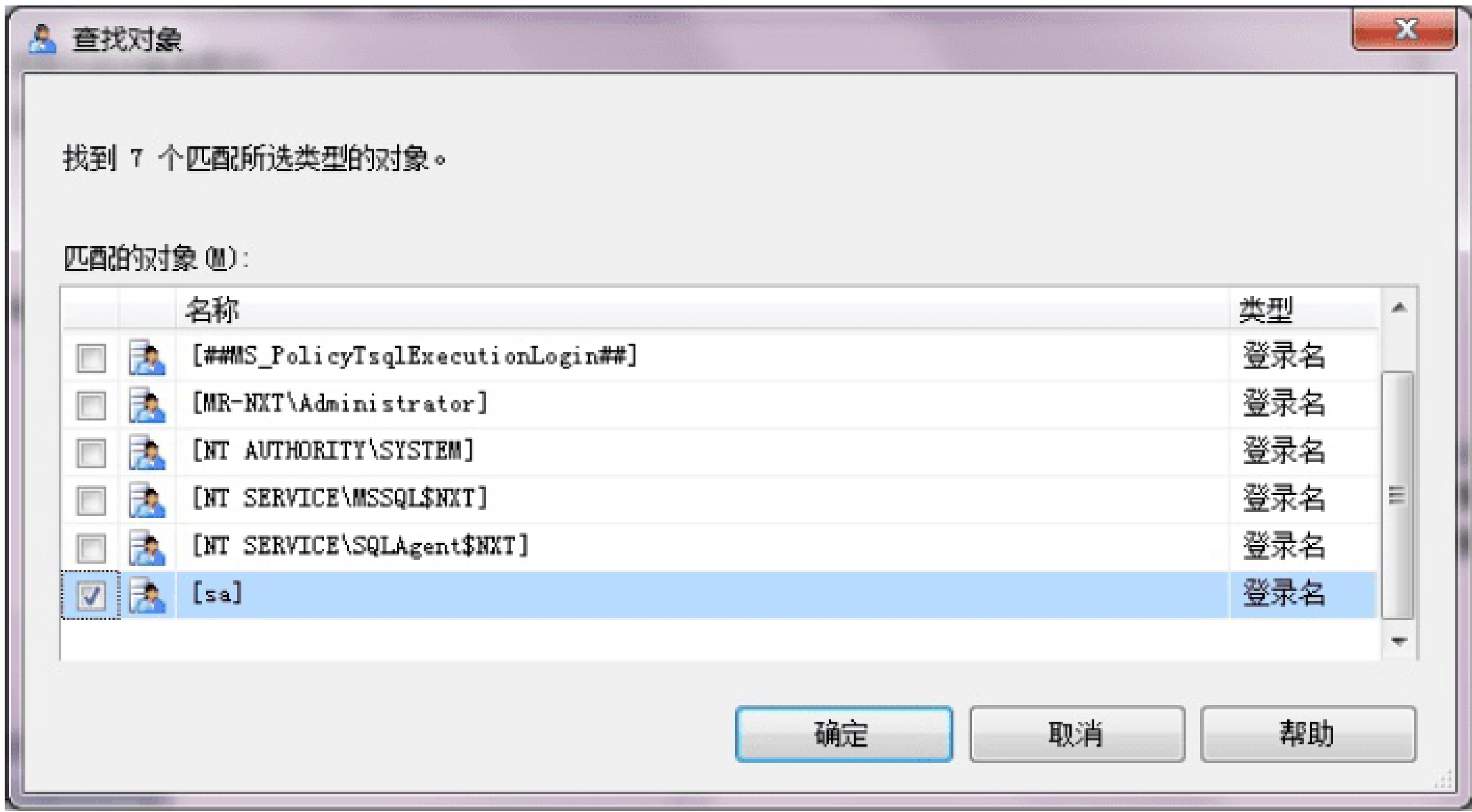


图 3.7　“查找对象”对话框

（6）在“匹配的对象”列表框中选择数据库的所有者 sa 选项，单击“确定”按钮，完成数据库所有者的更改操作。

2．使用 ALTER DATABASE 语句修改数据库

Transact-SQL 中修改数据库的命令为 ALTER DATABASE。其语法格式如下：

```
ALTER DATABASE database
{ADD FILE<filespec>[,…n][TO FILEGROUP filegroup_name]
|ADD LOG FILE<filespec>[,…n]
|REMOVE FILE logical_file_name
|ADD FILEGROUP filegroup_name
|REMOVE FILEGROUP filegroup_name
|MODIFY FILE<filespec>
|MODIFY NAME=new_dbname
|MODIFY FILEGROUP filegroup_name{filegroup_property|NAME=new_filegroup_name}
|SET<optionspec>[,…n][WITH<termination>]
```

```
|COLLATE<collation_name>
}
```

参数说明如下。

☑ ADD FILE：指定要添加的数据库文件。

☑ TO FILEGROUP：指定要添加文件到哪个文件组。

☑ ADD LOG FILE：指定要添加的事务日志文件。

☑ REMOVE FILE：从 SQL Server 的实例中删除逻辑文件说明并删除物理文件。除非文件为空，否则无法删除文件。

☑ ADD FILEGROUP：指定要添加的文件组。

☑ REMOVE FILEGROUP：从数据库中删除指定文件组的定义，并且删除其包含的所有数据库文件。文件组只有为空时才能被删除。

☑ MODIFY FILE：修改指定文件的文件名、容量大小、最大容量、文件增容方式等属性，但一次只能修改一个文件的一个属性。使用此选项时应注意，在文件格式 filespec 中必须用 NAME 明确指定文件名称，如果文件大小是已经确定的，那么新定义的 SIZE 必须比当前的文件容量大；FILENAME 只能指定在 tempdbdatabase 中存在的文件，并且新的文件名只有在 SQL Server 重新启动后才发生作用。

☑ MODIFY FILEGROUP filegroup_name filegroup_property：修改文件组属性，其中属性 filegroup_property 的取值可以为 READONLY，表示指定文件组为只读，要注意的是主文件组不能指定为只读，只有对数据库有独占访问权限的用户才可以将一个文件组标志为只读；取值为 READWRITE，表示使文件组为可读写，只有对数据库有独占访问权限的用户才可以将一个文件组标志为可读写；取值为 DEFAULT，表示指定文件组为默认文件组，一个数据库中只能有一个默认文件组。

☑ SET：设置数据库属性。

【例 3.01】 将一个大小为 10MB 的数据文件 mrkj 添加到 Mingri 数据库中，该数据文件的大小为 10MB，最大的文件大小为 100MB，增长速度为 2MB，Mingri 数据库的物理地址为 D 盘文件夹下。**（实例位置：资源包\源码\03\3.01）**

SQL 语句如下：

```
ALTER DATABASE Mingri
ADD FILE
(
NAME=mrkj,
Filename='D:\mrkj.ndf',
size=10MB,
Maxsize=100MB,
Filegrowth=2MB
)
```

3.3.3　删除数据库

DROP DATABASE 命令可以删除一个或多个数据库。当某一个数据库被删除后，这个数据库的所

有对象和数据都将被删除，所有日志文件和数据文件也都将删除，所占用的空间将会释放给操作系统。

1. 以界面方式删除数据库

下面介绍如何删除数据库 Mingri。具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要删除的数据库 Mingri 选项，在弹出的快捷菜单中选择“删除”命令，如图 3.8 所示。

（3）在弹出的“删除对象”窗口中单击“确定”按钮，即可删除数据库，如图 3.9 所示。

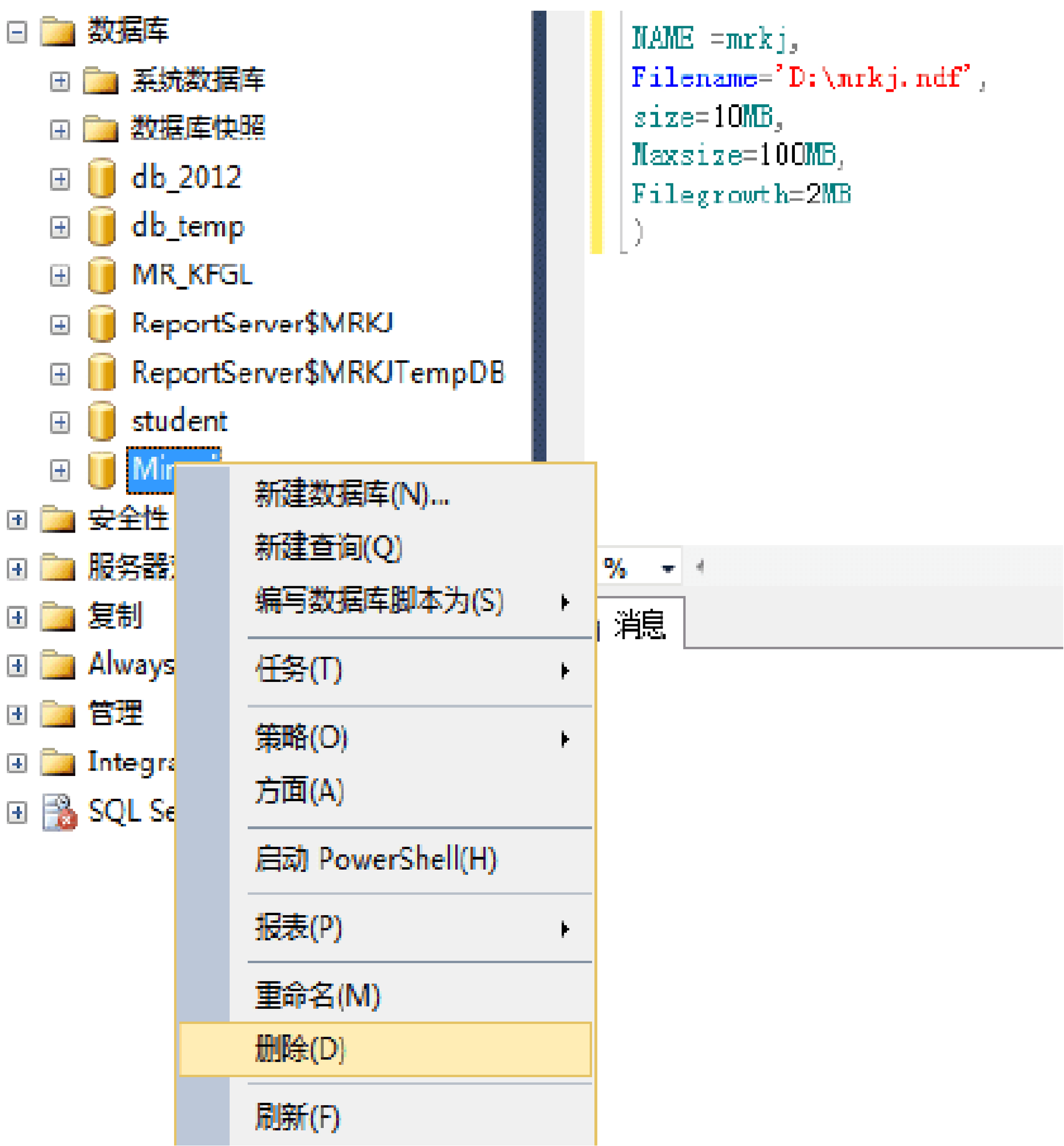


图 3.8　删除数据库

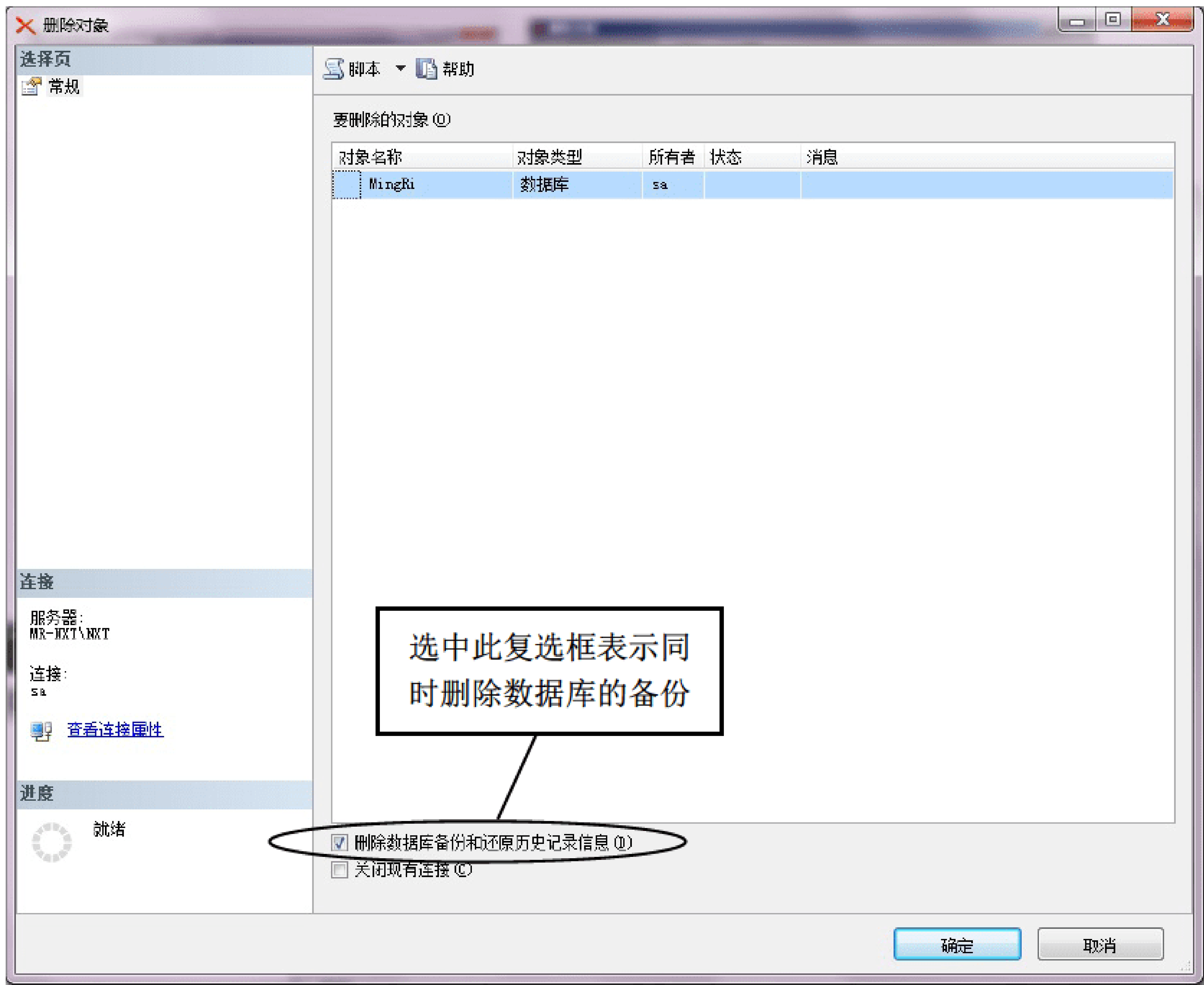


图 3.9　除去对象

注意

系统数据库（msdb、model、master、tempdb）无法删除。删除数据库后应立即备份 master 数据库，因为删除数据库将更新 master 数据库中的信息。

2．使用 DROP DATABASE 语句删除数据库

语法格式如下：

```
DROP DATABASE database_name [,...n]
```

其中，database_name 是要删除的数据库名称。


注意

使用 DROP DATABASE 命令删除数据库时，系统中必须存在所要删除的数据库，否则系统将会出现错误。

另外，如果删除正在使用的数据库，系统将会出现错误。

例如，不能在“学生档案管理”数据库中删除“学生档案管理”数据库，SQL 代码如下：

```
Use 学生档案管理                    --使用学生档案管理数据库
Drop database 学生档案管理   --删除正在使用的数据库
```

删除学生档案管理数据库的操作没有成功，系统会报错，运行结果如图 3.10 所示。

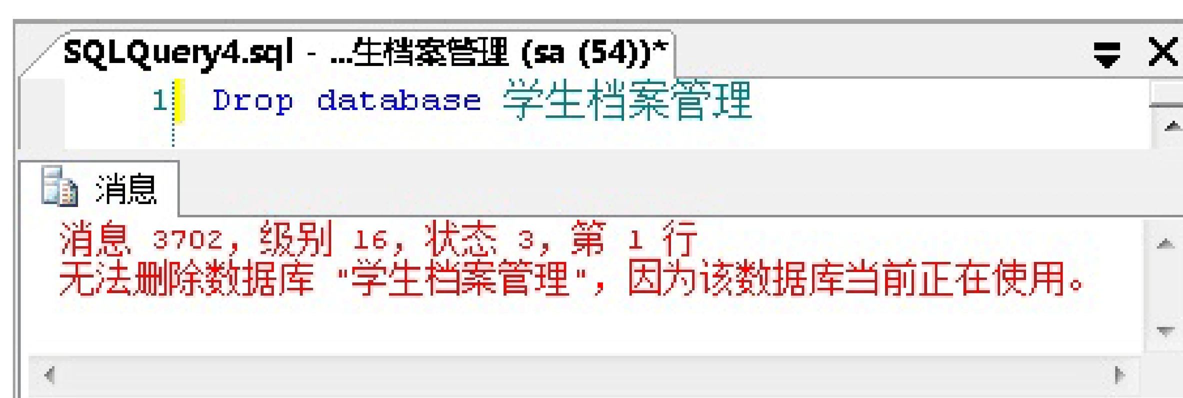


图 3.10　删除正在使用的数据库，系统会报错的效果图

在“学生档案管理”数据库中，使用 DROP DATABASE 命令删除数据库名为“学生档案管理”的数据库。

在查询编辑器窗口中的运行结果如图 3.11 所示。

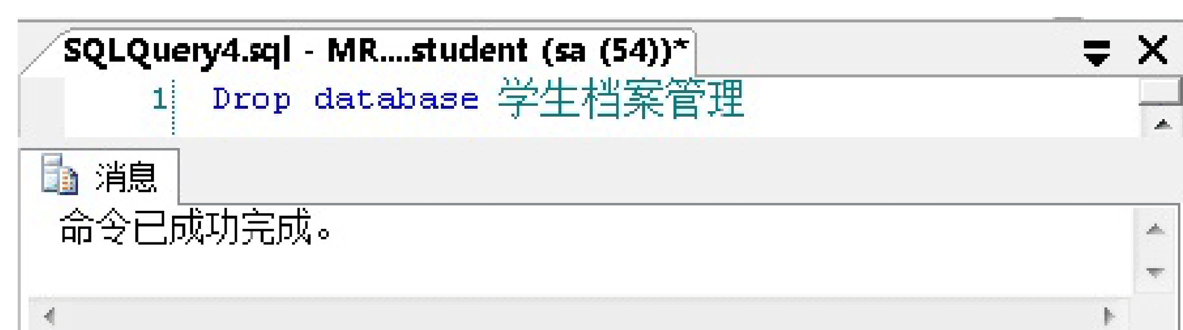


图 3.11　删除“学生档案管理”数据库

3.4　小　　结

本章介绍了 SQL Server 2014 数据库的组成、创建和管理数据库的方法以及如何查看数据库信息。读者不仅可以使用 SQL Server 2014 界面方式完成创建和管理数据库的工作，还可以调用 Transact-SQL 语句完成对应操作。

第 4 章

操作数据表

（视频讲解：60 分钟）

本章主要介绍使用 Transact-SQL 语句和使用 SQL Server Management Studio 创建数据表、修改数据表和删除数据表的过程。

学习摘要：

- 以界面方式创建表
- 以界面方式修改表
- 以界面方式删除表
- 使用 CREATE TABLE 语句创建表
- 使用 ALTER TABLE 语句修改表
- 使用 DROP TABLE 语句删除表

4.1　数据表的基础知识



表是最常见的一种组织数据的方式，一张表一般具有多个列（即多个字段）。每个字段都具有特定的属性，包括字段名、数据类型、字段长度、约束、默认值等，这些属性在创建表时被确定。

SQL Server 2014 提供了基本数据类型和自定义数据类型，下面分别对其进行介绍。

1．基本数据类型

基本数据类型按数据的表现方式及存储方式的不同可以分为整数数据类型、货币数据类型、浮点数据类型、日期/时间数据类型、字符数据类型、二进制数据类型、图像和文本数据类型以及 SQL Server 2014 引用的 3 种新数据类型。具体介绍如表 4.1 所示。

表 4.1　基本数据类型

分　类	数 据 特 性	数 据 类 型
整数数据类型	常用的一种数据类型，可以存储整数或者小数	BIT
		INT
		SMALLINT
		TINYINT
货币数据类型	用于存储货币值，使用时在数据前加上货币符号，不加货币符号的情况下默认为“￥”	MONEY
		SMALLMONEY
浮点数据类型	用于存储十进制小数	REAL
		FLOAT
		DECIMAL
		NUMERIC
日期/时间数据类型	用于存储日期类型和时间类型的组合数据	DATETIME
		SMALLDATETIME
		DATA
		DATETIME(2)
		DATETIMESTAMPOFFSET
字符数据类型	用于存储各种字母、数字符号和特殊符号	CHAR
		NCHAR(n)
		VARCHAR
		NVARCHAR(n)
二进制数据类型	用于存储二进制数据	BINARY
		VARBINARY
图像和文本数据类型	用于存储大量的字符及二进制数据（Binary Data）	TEXT
		NTEXT(n)
		IMAGE

2. 用户自定义数据类型

用户自定义数据类型并不是真正的数据类型，它只是提供了一种加强数据库内部元素和基本数据类型之间一致性的机制。通过使用用户自定义数据类型，能够简化对常用规则和默认值的管理。

在 SQL Server 2014 中，创建用户自定义数据类型有两种方法：一是使用界面方式，二是使用 SQL 语句，下面分别介绍。

（1）使用界面方式创建用户定义数据类型

在 db_database 数据库中，创建用来存储邮政编码信息的 postcode 用户定义数据类型，数据类型为 char，长度为 8000。

操作步骤如下。

① 选择“开始”→“所有程序”→Microsoft SQL Server 2014→SQL Server Management Studio 命令，打开 SQL Server 2014。

② 在 SQL Server 2014 的“对象资源管理器”中，依次展开“数据库”→“选择指定数据库”→“可编程性”→“类型”的节点。

③ 展开“类型”节点，选中“用户定义数据类型”，单击鼠标右键，在弹出的快捷菜单中选择“新建用户定义数据类型”命令。在打开的窗口中设置用户定义数据类型的名称、依据的系统数据类型以及是否允许 NULL 值等，如图 4.1 所示，还可以将已创建的规则和默认值绑定到该用户定义的数据类型上。

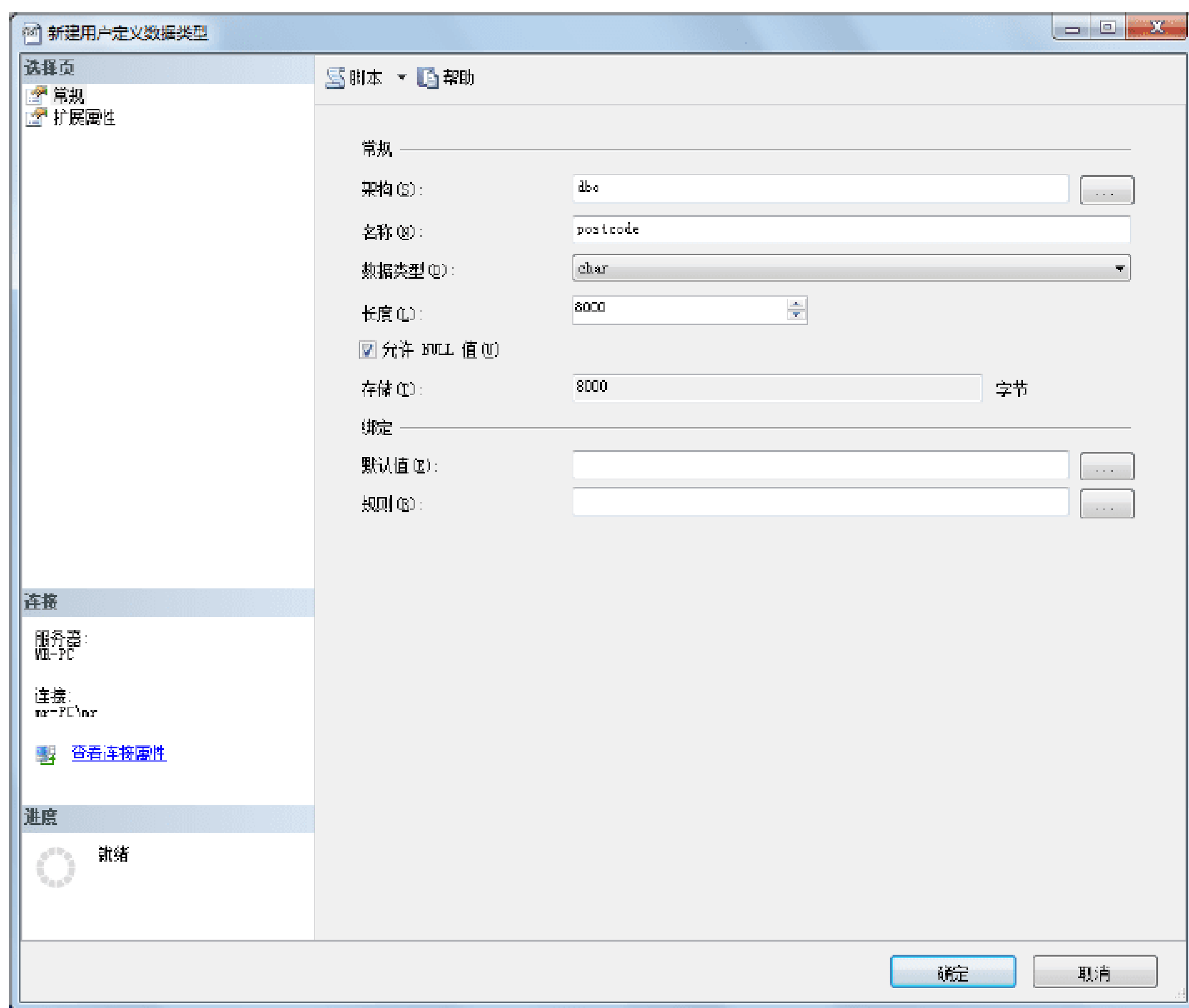


图 4.1　创建用户自定义数据类型

④ 单击“确定”按钮，完成创建工作。

（2）使用 SQL 语句创建用户自定义数据类型

在 SQL Server 2014 中，使用系统数据类型 sp_addtype 创建用户自定义数据类型。

语法格式如下：

```
sp_addtype[@typename=]type,
[@phystype=]system_data_type
[,[@nulltype=]'null_type']
[,[@owner=]'owner_name']
```

参数说明如下。

- ☑ [@typename=]type：指定待创建的用户自定义数据类型的名称。用户定义数据类型名称必须遵循标识符的命名规则，而且在数据库中唯一。
- ☑ [@phystype=]system_data_type：指定用户定义数据类型所依赖的系统数据类型。
- ☑ [@nulltype=]'null_type'：指定用户定义数据类型的可空属性，即用户定义数据类型处理空值的方式。取值为 NULL、NOT NULL 或 NONULL。

在 db_database 数据库中，创建用来存储邮政编码信息的 postcode 用户自定义数据类型。在查询编辑器窗口中运行的结果如图 4.2 所示。

SQL 语句如下：

```
USE db_database
EXEC sp_addtype postcode,'char(8)','not null'
```

创建用户定义数据类型后，就可以像系统数据类型一样使用用户自定义数据类型。例如，在 db_database 数据库的 tb_Student 表中创建新的字段，为字段“邮政编码”指定数据类型时，就可以在下拉列表框中选择刚刚创建的用户数据类型 postalcode 了，如图 4.3 所示。

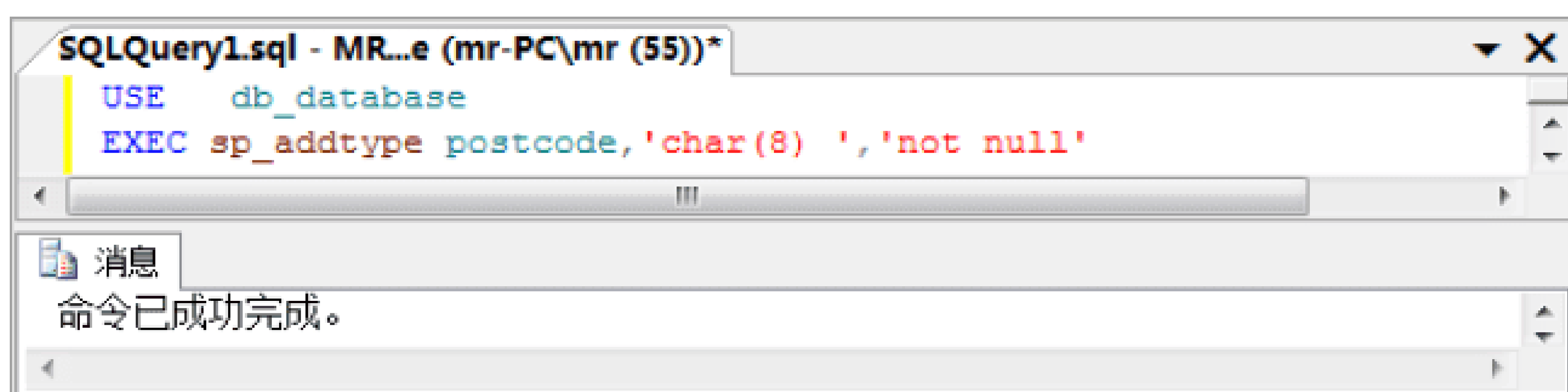


图 4.2　用户自定义 postalcode 类型

学生编号	int	☐
学生姓名	nvarchar(50)	☑
性别	nvarchar(50)	☑
出生年月	smalldatetime	☑
年龄	int	☑
所在学院	nvarchar(50)	☑
所学专业	nvarchar(50)	☑
家庭住址	nvarchar(50)	☑
统招否	bit	☐
备注信息	nvarchar(50)	☑
邮政编码	postalcode:char(8)	☑

图 4.3　创建字段时用了 postalcode 数据类型

根据需要，还可以修改、删除用户数据类型。SQL Server 2014 提供系统存储过程 sp_droptype，该存储过程从 systypes 删除别名数据类型。

3．数据表的数据完整性

表列中除了具有数据类型和大小属性之外，还有其他属性。其他属性是保证数据库中数据完整性和表的引用完整性的重要部分。

数据完整性是指列中每个事件都有正确的数据值。数据值的数据类型必须正确，并且数据值必须位于正确的域中。

引用完整性指示表之间的关系得到正确维护。一个表中的数据只应指向另一个表中的现有行，不应指向不存在的行。

SQL Server 2014 提供多种强制数据完整性的机制。下面分别对其进行介绍。

（1）空值与非空值（NULL 或 NOT NULL）

表的每一列都有一组属性，如名称、数据类型、数据长度和为空性等，列的所有属性即构成列的定义。列可以定义为允许或不允许空值。

☑ 允许空值（NULL）：默认情况下，列允许空值，即允许用户在添加数据时省略该列的值。

☑ 不允许空值（NOT NULL）：不允许在没有指定列默认值的情况下省略该列的值。

（2）默认值

如果在插入行时没有指定列的值，那么默认值将指定列中所使用的值。默认值可以是任何取值为常量的对象，如内置函数和数学表达式等。下面介绍两种使用默认值的方法。

在 CREATE TABLE 中使用 DEFAULT 关键字创建默认定义，将常量表达式指派为列的默认值，这是标准方法。

使用 CREATE DEFAULT 语句创建默认对象，然后使用 sp_bindefault 系统存储过程将它绑定到列上，这是一个向前兼容的功能。

（3）特定标识属性（IDENTITY）

数据表中如果某列被指派特定标识属性（IDENTITY），系统将自动为表中插入的新行生成连续递增的编号。因为标识值通常唯一，所以标识列常定义为主键。

IDENTITY 属性适用于 INT、SMALLINT、TINYINT、DECIMAL(P,0)、NUMERIC(P,0)数据类型的列。



注意

一个列不能同时具有 NULL 属性和 IDENTITY 属性，二者只能选其一。

（4）约束

约束是用来定义 SQL Server 2014 自动强制数据库完整性的方式。使用约束优先于使用触发器、规则和默认值。SQL Server 2014 中共有以下 5 种约束。

① 非空（NOT NULL）：使用户必须在表的指定列中输入一个值。每个表中可以有多个非空约束。

② 检查（CHECK）：用来指定一个布尔操作，限制输入到表中的值。

③ 唯一性（UNIQUE）：使用户的应用程序必须向列中输入一个唯一的值，值不能重复，但可以为空。

④ 主键（PRIMARY KEY）：建立一列或多列的组合以唯一标识表中的每一行。主键可以保证实体完整性，一个表只能有一个主键，同时主键中的列不能接受空值。

⑤ 外键（FOREIGN KEY）：外键是用于建立和加强两个表数据之间的链接的一列或多列。当一个表中作为主键的一列被添加到另一个表中时，链接就建立了，主要目的是控制存储在外键表中的数据。

4.2 表的设计原则

数据库中的表与人们在日常生活中使用的表格类似。数据库中的表也是由行和列组成的。相同类

的信息组成了列，每一列又称为一个字段，每列的列标题称为字段名。在每一行中，包含了许多列的信息，每一行数据称为一条记录。一个数据表是由一条或多条记录组成的，没有记录的表称为空表。

在设计数据库时，应该先确定需要什么样的表，各表中都有哪些数据，以及各个表的存取权限等。

创建表的最有效的方法是将表中所需的信息一次定义完成，也可以先创建一个表，然后再向其填入数据。

设计表时应注意下列问题。

（1）表中包含的数据类型。

（2）表的各列及每一列的数据类型。

（3）哪些列允许空值。

（4）是否要使用以及何时使用约束、默认设置或规则。

（5）所需索引的类型，哪里需要索引，哪些列是主键，哪些是外键。

在创建表时必须满足以下规定。

（1）每个表有一个名称，称为表名或关系名。表名必须以字母开头，最大长度为 30 个字符。

（2）一张表中可以包含若干个列，但是列名必须唯一。列名也称为属性名。

（3）同一列中的数据必须要有相同的数据类型。

（4）表中的每一列数值必须为一个不可分割的数据项。

（5）表中的一行称为一条记录。

4.3　以界面方式创建、修改和删除数据表



4.3.1　创建数据表

下面在 SQL Server Management Studio 中创建数据表 mrkj，具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）鼠标右键单击“表”选项，在弹出的快捷菜单中选择“新建表”命令，如图 4.4 所示。

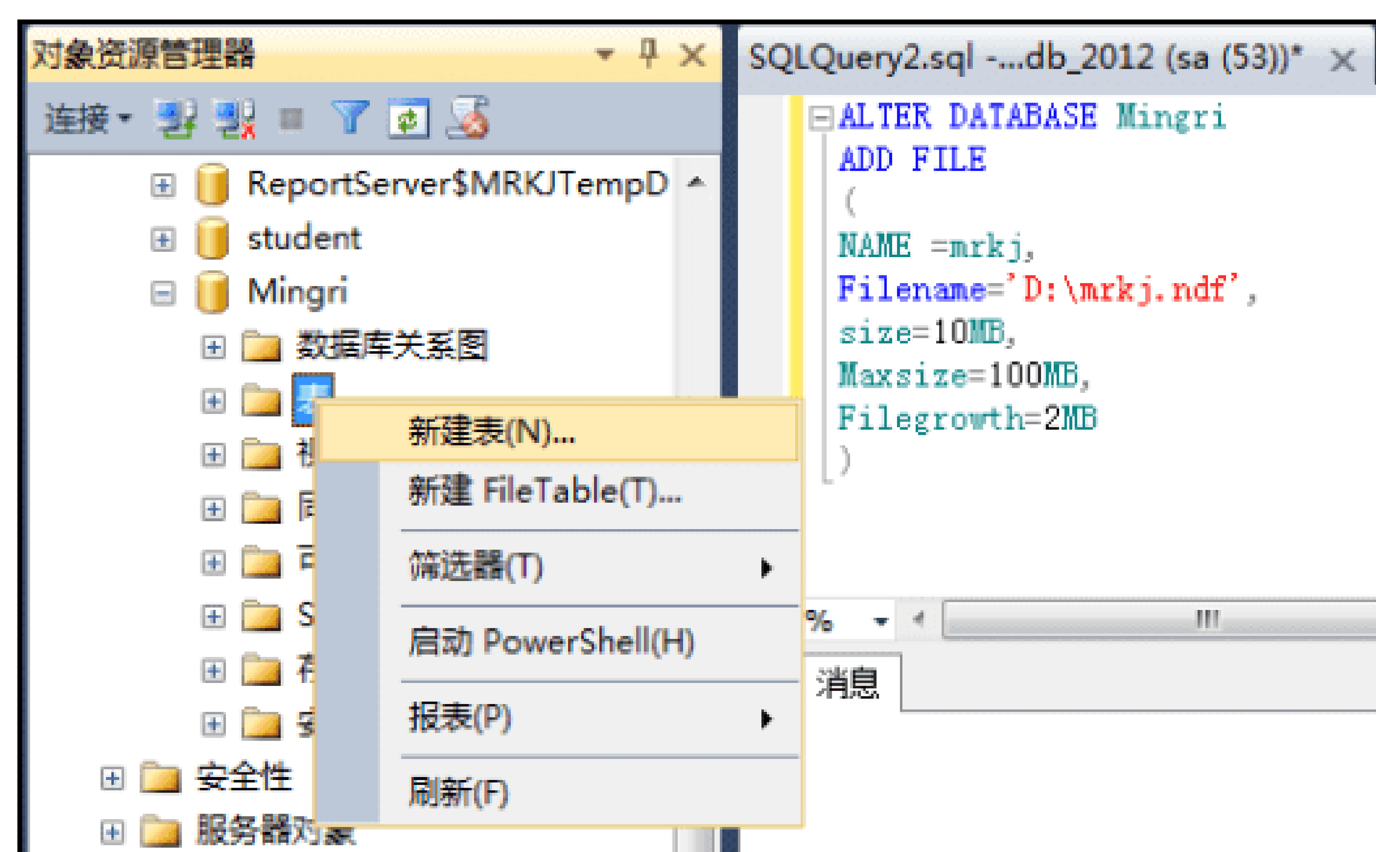


图 4.4　新建表

（3）进入表设计窗口，如图 4.5 所示。在列表框中填写所需要的字段名，单击“保存”按钮，即添加表成功。

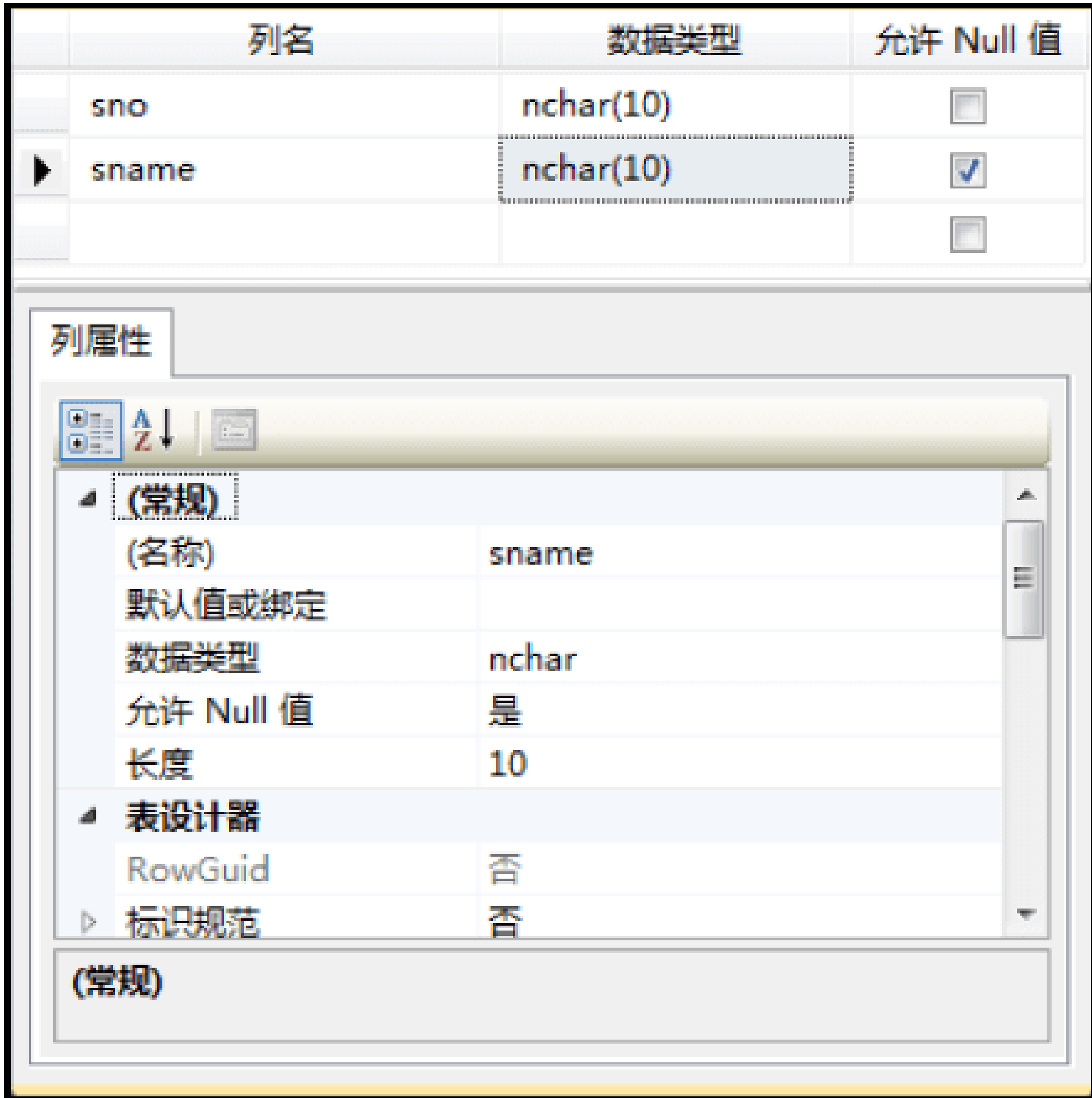


图 4.5　创建数据表名称

4.3.2　修改数据表

下面介绍如何更改表 mrkj 的所有者。具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库，在“对象资源管理器”中展开“数据库”下面的表节点。

（2）鼠标右键单击需要更改的表 mrkj，在弹出的快捷菜单中选择“设计”命令，如图 4.6 所示。

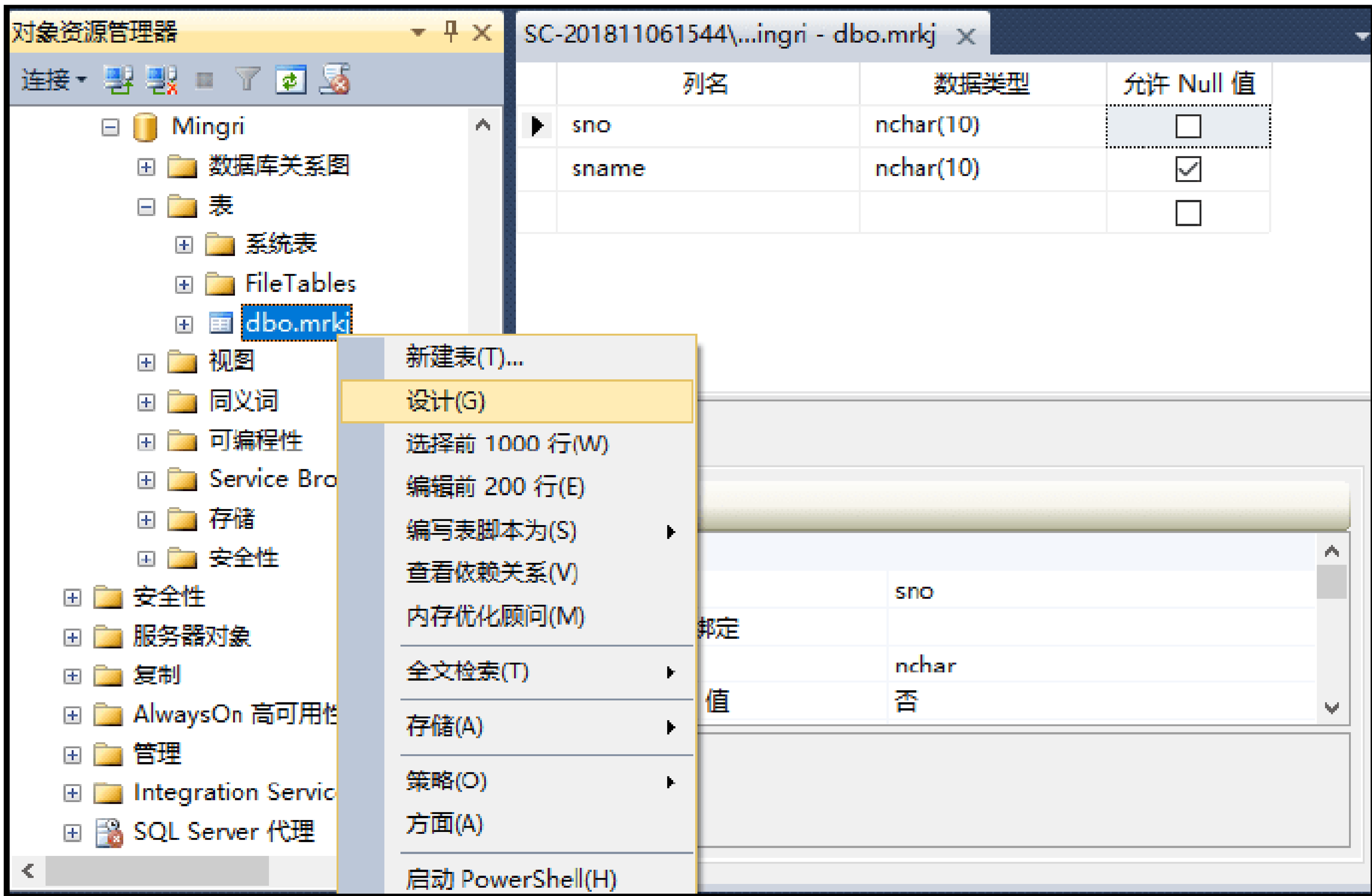


图 4.6　选择“设计”命令

（3）进入表设计窗口，如图 4.7 所示。通过该窗口可以修改数据表的相关选项。修改完成后，单击“保存”按钮，修改成功。

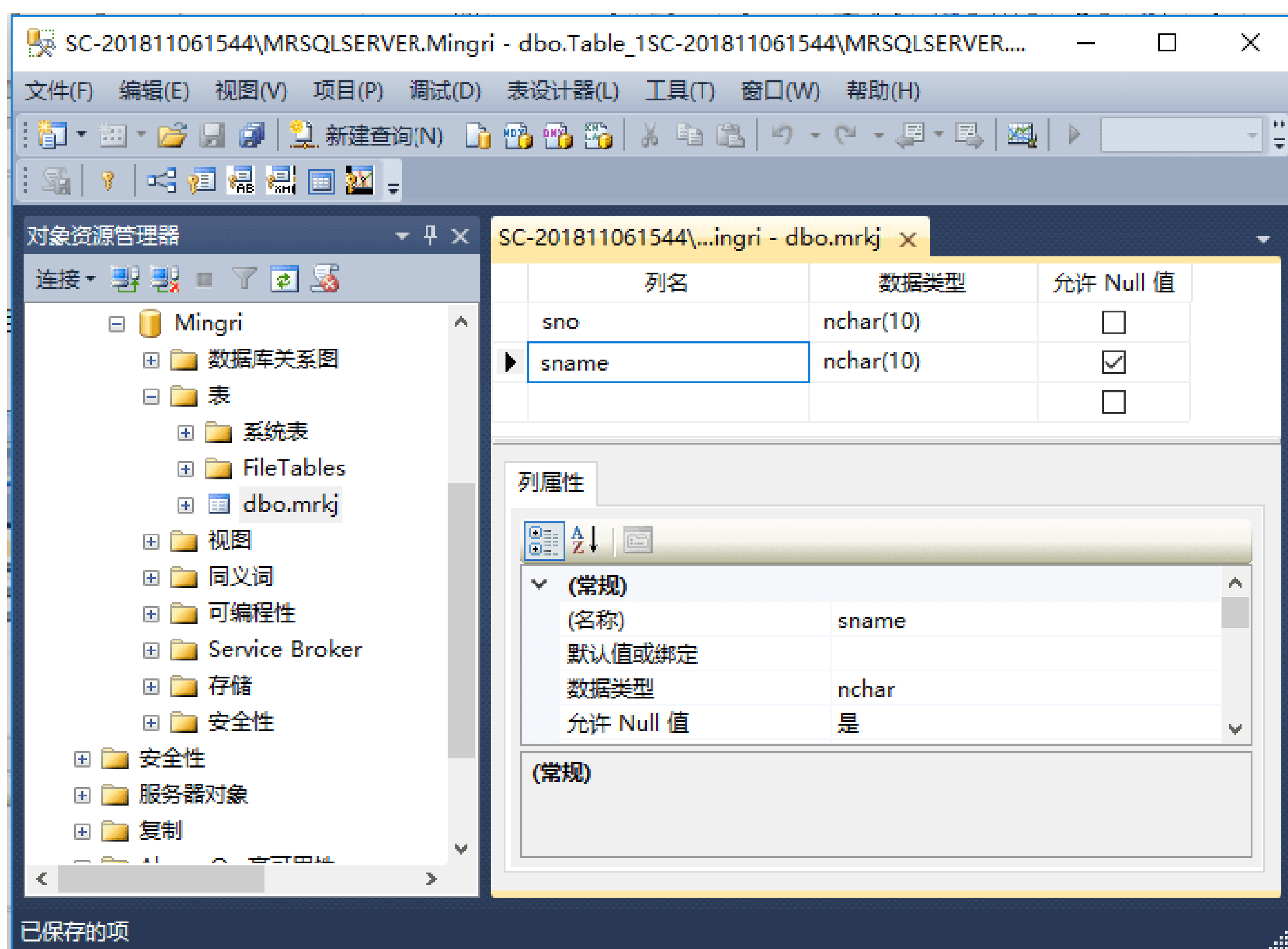


图 4.7　修改表字段

4.3.3　删除数据表

下面介绍如何删除表 mrkj 的所有者。具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库，在“对象资源管理器”中展开“数据库”下面的表节点。

（2）鼠标右键单击需要删除的表 mrkj，在弹出的快捷菜单中选择“删除”命令，如图 4.8 所示。

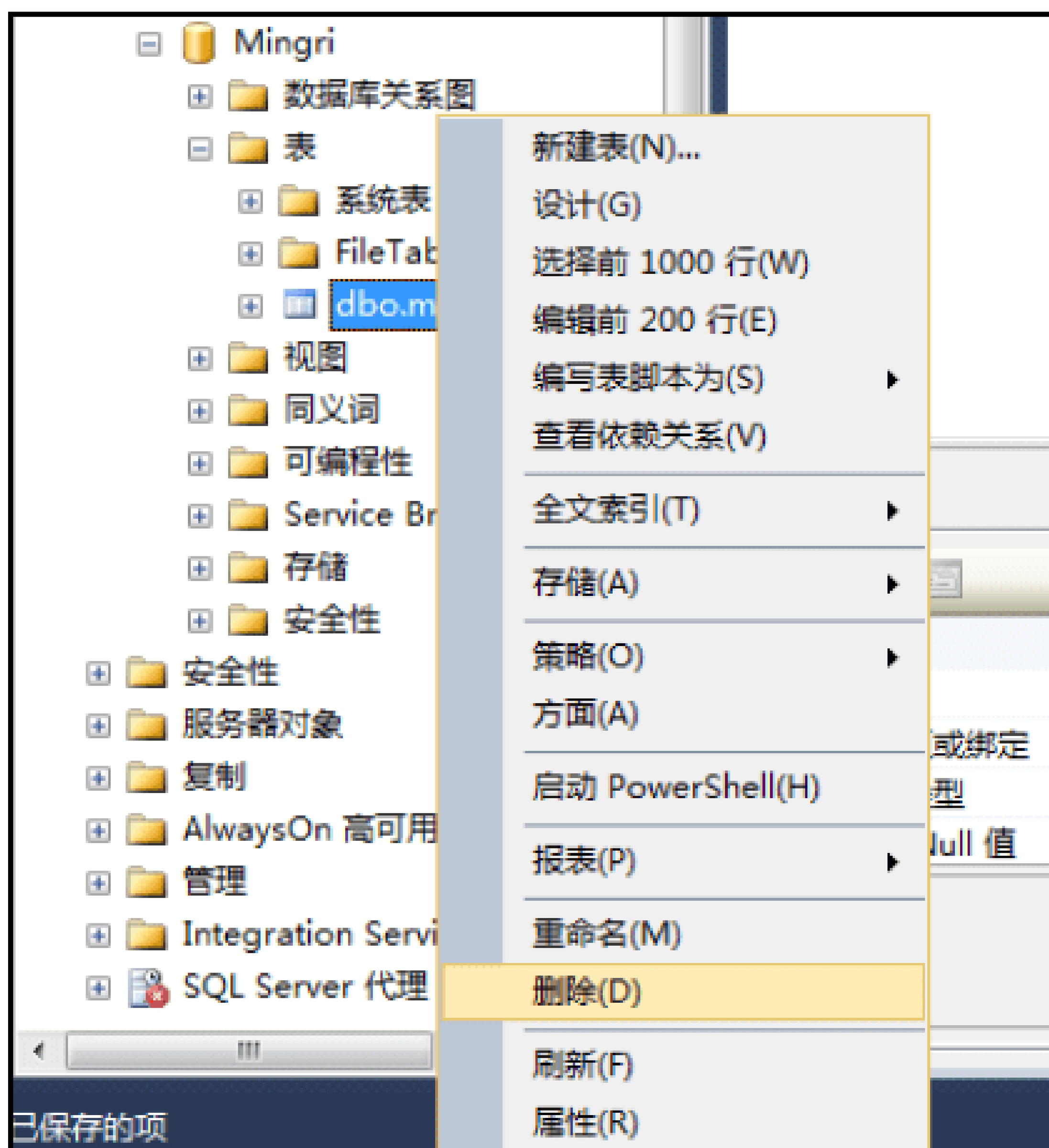


图 4.8　选择表删除

（3）打开“删除对象”窗口，如图 4.9 所示。通过该窗口可以删除数据表的相关选项。单击“确定”按钮，删除成功。

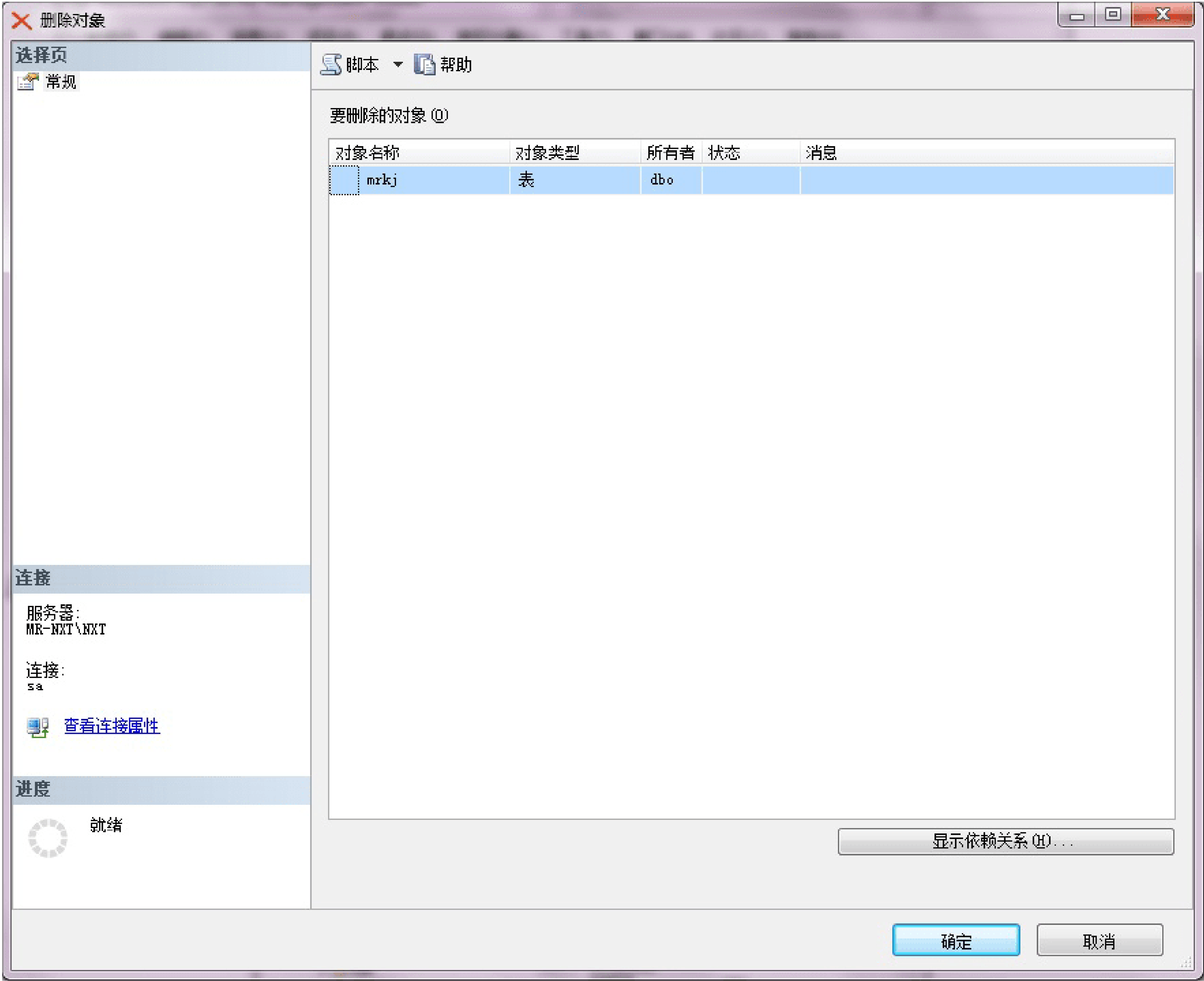


图 4.9　删除表



4.4 创 建 表

使用 CREATE TABLE 语句可以创建表，其基本语法格式如下：

```
CREATE TABLE
[database_name.[owner] .| owner.] table_name
({<column_definition>
| column_name AS computed_column_expression
| <table_constraint> ::= [CONSTRAINT constraint_name]}
| [{PRIMARY KEY | UNIQUE} [,...n]]
)
[ON {filegroup | DEFAULT}]
[TEXTIMAGE_ON {filegroup | DEFAULT}]
<column_definition> ::= {column_name data_type}
[COLLATE <collation_name>]
[[DEFAULT constant_expression]
| [IDENTITY [(seed , increment) [NOT FOR REPLICATION]]]
]
[ROWGUIDCOL]
```

```
[<column_constraint>] [...n]
<column_constraint> ::= [CONSTRAINT constraint_name]
{[NULL | NOT NULL]
| [{PRIMARY KEY | UNIQUE}
[CLUSTERED | NONCLUSTERED]
[WITH FILLFACTOR = fillfactor]
[ON {filegroup | DEFAULT}]]
]
| [[FOREIGN KEY]
REFERENCES ref_table [(ref_column)]
[ON DELETE {CASCADE | NO ACTION}]
[ON UPDATE {CASCADE | NO ACTION}]
[NOT FOR REPLICATION]
]
| CHECK [NOT FOR REPLICATION]
(logical_expression)
}
<table_constraint> ::= [CONSTRAINT constraint_name]
{[{PRIMARY KEY | UNIQUE}
[CLUSTERED | NONCLUSTERED]
{(column [ASC | DESC] [,...n])}
[WITH FILLFACTOR = fillfactor]
[ON {filegroup | DEFAULT}]
]
| FOREIGN KEY
[(column [,...n])]
REFERENCES ref_table [(ref_column [,...n])]
[ON DELETE {CASCADE | NO ACTION}]
[ON UPDATE {CASCADE | NO ACTION}]
[NOT FOR REPLICATION]
| CHECK [NOT FOR REPLICATION]
(search_conditions)
}
```

CREATE TABLE 语句的参数及说明如表 4.2 所示。

表 4.2　CREATE TABLE 语句的参数及说明

参　　数	描　　述
database_name	在其中创建表的数据库的名称。database_name 必须指定现有数据库的名称。如果未指定，则 database_name 默认为当前数据库
owner	新表所属架构的名称
table_name	新表的名称。表名必须遵循标识符规则。除了本地临时表名（以单个数字符号（#）为前缀的名称）不能超过 116 个字符外，table_name 最多可包含 128 个字符
column_name	表中列的名称。列名必须遵循标识符规则，并且在表中是唯一的
computed_column_expression	定义计算列的值的表达式

续表

参　　数	描　　述
ON{filegroup|default}	指定存储表的分区架构或文件组
<table_constraint>	表约束
TEXTIMAGE_ON{filegroup|"default"}}	指定 text、ntext、image、xml、varchar(max)、nvarchar(max)、varbinary(max)列存储在指定文件组的关键字
CONSTRAINT	可选关键字，表示 PRIMARY KEY、NOT NULL、UNIQUE、FOREIGN KEY 或 CHECK 约束定义的开始
constraint_name	约束的名称。约束名称必须在表所属的架构中唯一
NULL | NOT NULL	确定列中是否允许使用空值
PRIMARY KEY	是通过唯一索引对给定的一列或多列强制实体完整性的约束。每个表只能创建一个 PRIMARY KEY 约束
UNIQUE	一个约束，该约束通过唯一索引为一个或多个指定列提供实体完整性。一个表可以有多个 UNIQUE 约束
CLUSTERED | NONCLUSTERED	指示为 PRIMARY KEY 或 UNIQUE 约束创建聚集索引还是非聚集索引。PRIMARY KEY 约束默认为 CLUSTERED，UNIQUE 约束默认为 NONCLUSTERED
column	用括号括起来的一列或多列，在表约束中表示这些列用在约束定义中
[ASC | DESC]	指定加入到表约束中的一列或多列的排序顺序。默认值为 ASC
WITH FILLFACTOR = fillfactor	指定数据库引擎存储索引数据时每个索引页的填充满程度。用户指定的 fillfactor 值可以为 1～100 之间的任意值。如果未指定值，则默认值为 0
FOREIGN KEY REFERENCES	为列中的数据提供引用完整性的约束。FOREIGN KEY 约束要求列中的每个值在所引用的表中对应的被引用列中都存在
(ref_column [,... n])	是 FOREIGN KEY 约束所引用的表中的一列或多列
ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}	指定如果已创建表中的行具有引用关系，并且被引用行已从父表中删除，则对这些行采取的操作。默认值为 NO ACTION
ON UPDATE {NO ACTION | CASCADE }	指定在发生更改的表中，如果行有引用关系且引用的行在父表中被更新，则对这些行采取什么操作。默认值为 NO ACTION
CHECK	一个约束，该约束通过限制可输入一列或多列中的可能值来强制实现域完整性。计算列上的 CHECK 约束也必须标记为 PERSISTED
NOT FOR REPLICATION	在 CREATE TABLE 语句中，可为 IDENTITY 属性、FOREIGN KEY 约束和 CHECK 约束指定 NOT FOR REPLICATION 子句

【例 4.01】　创建员工基本信息表。（**实例位置：资源包\源码\04\4.01**）

员工信息表（tb_basicMessage）：id 字段为 int 类型并且不允许为空；name 字段是长度为 10 的 varchar 类型；age 字段为 int 类型，dept 字段为 int 类型，headship 字段为 int 类型，SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[tb_basicMessage](
[id] [int] NOT NULL,
[name] [varchar](10),
```

```
[age] [int],
[dept] [int],
[headship] [int]
)
```

4.5　创建、修改和删除约束


视频讲解

4.5.1　非空约束

列为空性决定表中的行是否可为该列包含空值。空值（或 NULL）不同于零（0）、空白或长度为零的字符串（如""）。NULL 的意思是没有输入。出现 NULL 通常表示值未知或未定义。

1．创建非空约束

以界面方式创建非空约束的操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）鼠标右键单击要创建约束的表，在弹出的快捷菜单中选择“设计”命令，如图 4.10 所示。

（4）在表设计窗口中选中数据表中的“允许 Null 值”列，可以将指定的数据列设置为允许空或不允许空，将复选框选中便将该列设置为允许空。或者在列属性中在“允许 Null 值”的下拉列表框中选择“是”或“否”，选择“是”便将该列设置为允许空，如图 4.11 所示。

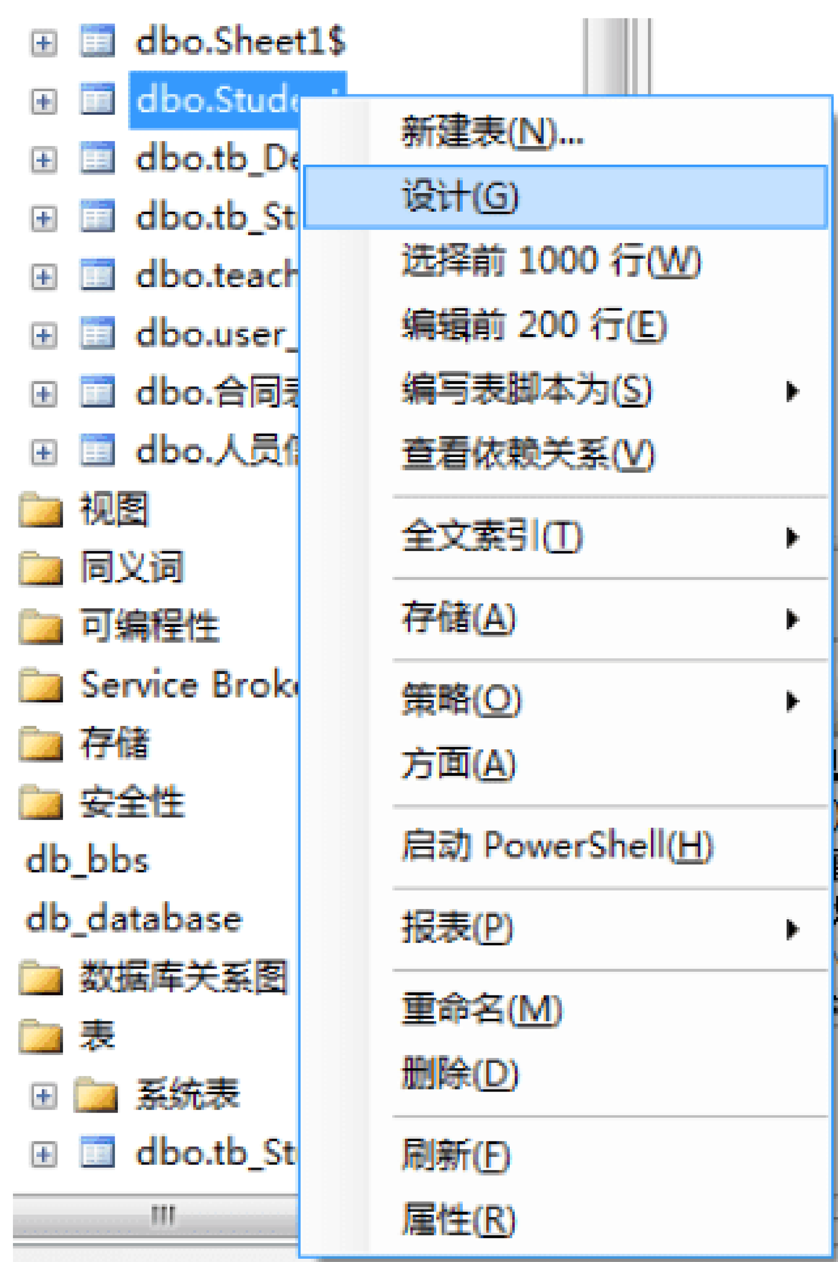


图 4.10　选择“设计”命令

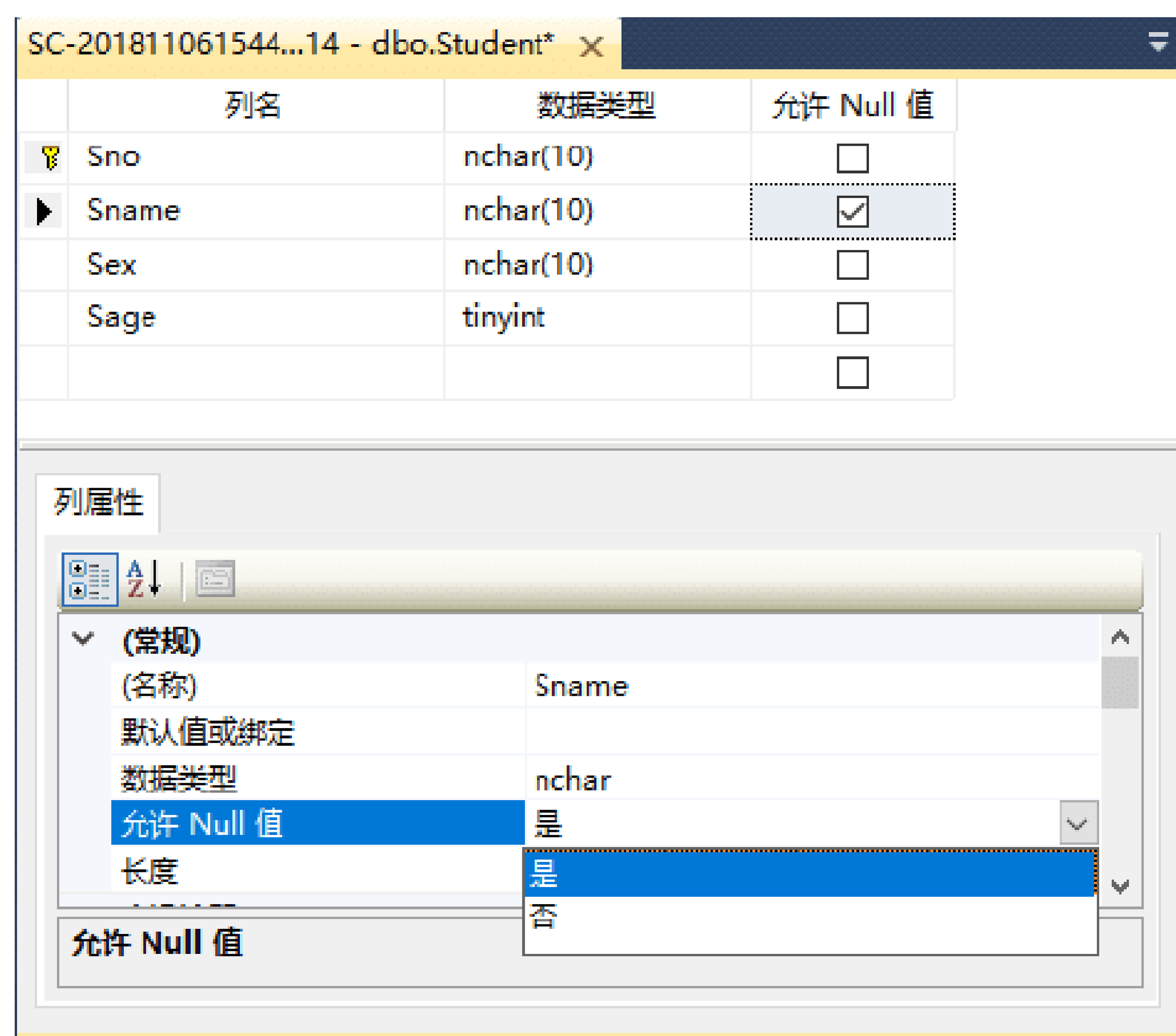


图 4.11　设定非空约束

可以在 CREATE TABLE 创建表时，使用 NOT NULL 关键字指定非空约束，其语法格式如下：

```
[CONSTRAINT <约束名>] NOT NULL
```

在例 4.01 中，通过使用 NOT NULL 关键字指定 id 字段不允许空。

2．修改非空约束

修改非空约束的语法格式如下：

```
ALTER TABLE table_name
ALTER COLUMN column_name column_type NULL | NOT NULL
```

参数说明如下。

- ☑ table_name：要修改非空约束的表名称。
- ☑ column_name：要修改非空约束的列名称。
- ☑ column_type：要修改非空约束的类型。
- ☑ NULL | NOT NULL：修改为空或者非空。

【例 4.02】 修改 tb_Student 表中的非空约束。（**实例位置：资源包\源码\04\4.02**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE tb_Student
ALTER COLUMN ID int NULL
```

3．删除非空约束

若要删除非空约束，将“允许 Null 值”复选框的选中状态取消即可。或者将“列属性”中的“允许 Null 值”设置为“否”，单击按钮，将修改后的表保存。

4.5.2 主键约束

可以通过定义 PRIMARY KEY 约束来创建主键，用于强制表的实体完整性。一个表只能有一个 PRIMARY KEY 约束，并且 PRIMARY KEY 约束中的列不能接受空值。由于 PRIMARY KEY 约束可保证数据的唯一性，因此经常对标识列定义这种约束。

1．创建主键约束

以界面方式创建主键约束的操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）鼠标右键单击要创建约束的表，在弹出的快捷菜单中选择“设计”命令。

（4）在弹出的表设计窗口中选择要设置为主键的列，可以通过快捷工具栏中的按钮进行单一设定，还可以将列选择多个，并通过单击鼠标右键，选择“设置主键”命令，将一个或多个列设置为主键，如图 4.12 所示。

（5）设置完成后，单击快键工具栏中的按钮保存主键设置，并关闭此窗体。

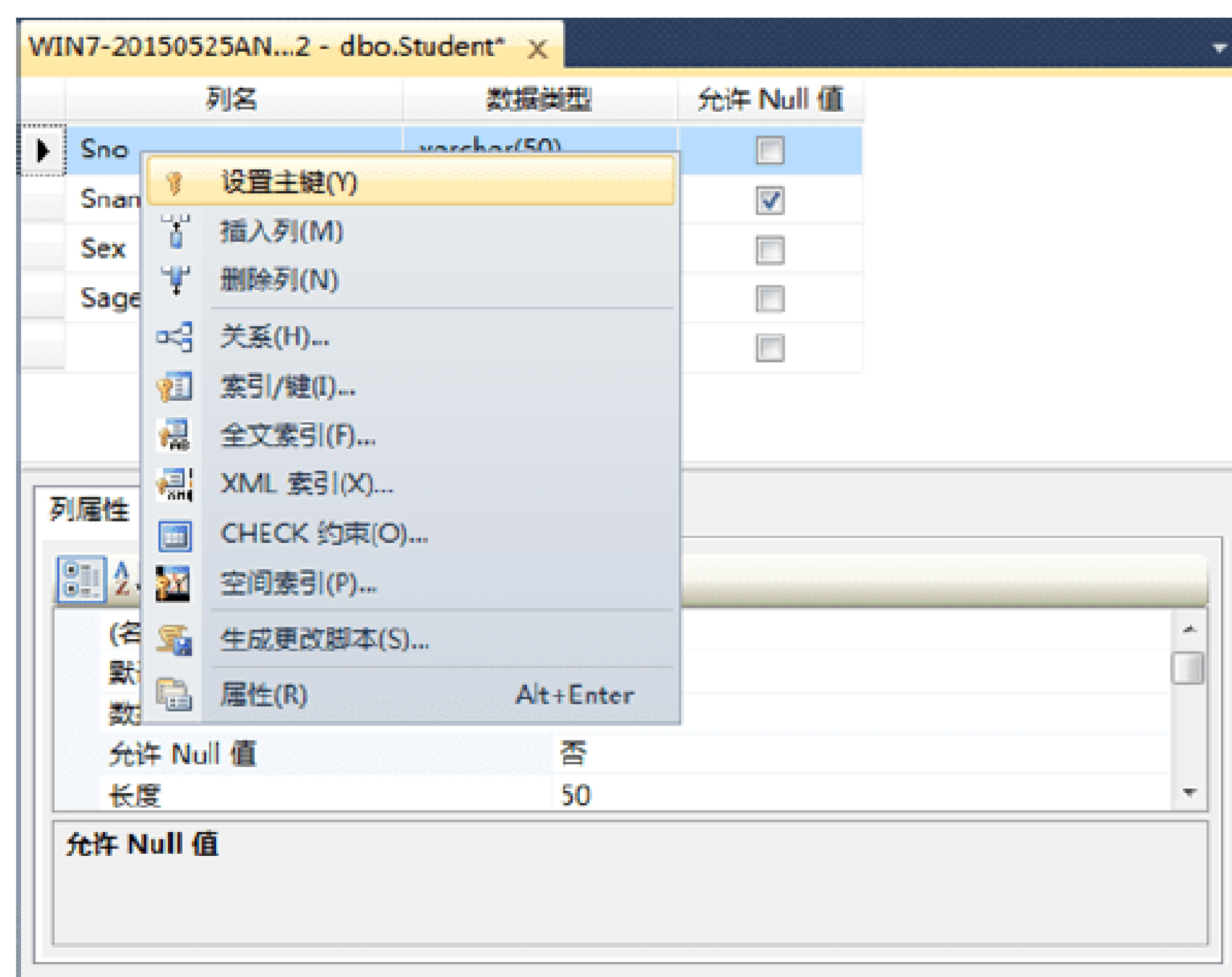


图 4.12　将多个列设置为主键



注意

将某列设置为主键时，不可以将此列设置为允许空，否则将弹出如图 4.13 所示的信息框，也不允许有重复的值。

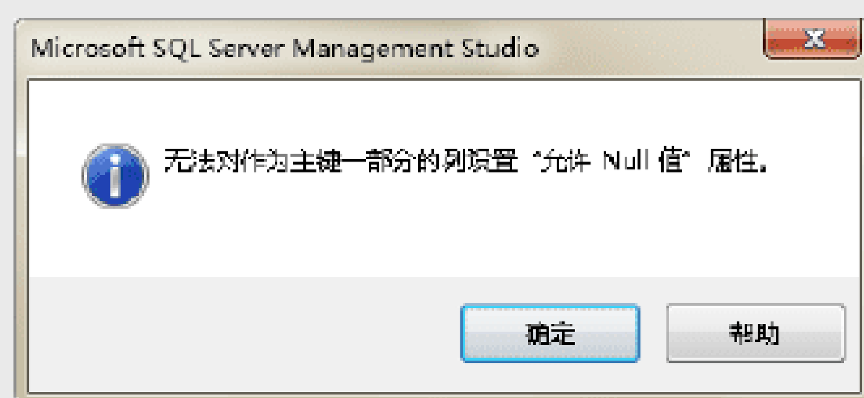


图 4.13　主键设置错误提示对话框

用 SQL 语句创建主键约束如下。

（1）在创建表时创建主键约束

【例 4.03】 创建数据表 Employee，并将字段 ID 设置主键约束。（**实例位置：资源包\源码\04\4.03**）

SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[Employee](
[ID] [int] CONSTRAINT PK_ID PRIMARY KEY,
[Name] [char](50),
[Sex] [char](2),
[Age] [int]
)
```



说明

在上述的语句中， CONSTRAINT PK_ID PRIMARY KEY 为创建一个主键约束，PK_ID 为用户自定义的主键约束名称，主键约束名称必须是合法的标识符。

（2）在现有表中创建主键约束

以 SQL 语句方式在现有表中创建主键约束的语法格式如下：

```
ALTER TABLE table_name
ADD
CONSTRAINT constraint_name
PRIMARY KEY [CLUSTERED | NONCLUSTERED]
{(Column[,…n])}
```

参数说明如下。

- ☑ CONSTRAINT：创建约束的关键字。
- ☑ constraint_name：创建约束的名称。
- ☑ PRIMARY KEY：表示所创建约束的类型为主键约束。
- ☑ CLUSTERED | NONCLUSTERED：是表示为 PRIMARY KEY 或 UNIQUE 约束创建聚集或非聚集索引的关键字。PRIMARY KEY 约束默认为 CLUSTERED，UNIQUE 约束默认为 NONCLUSTERED。

【例 4.04】 将 tb_Student 表中的 ID 字段指定设置主键约束。（**实例位置：资源包\源码\04\4.04**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE tb_Student
ADD CONSTRAINT PRM_ID PRIMARY KEY (ID)
```

2. 修改主键约束

若要修改 PRIMARY KEY 约束，必须先删除现有的 PRIMARY KEY 约束，然后再新定义重新创建该约束。

3. 删除主键约束

在界面中删除主键约束的步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）鼠标右键单击要创建约束的表，在弹出的快捷菜单中选择“设计”命令。

（4）在弹出的表设计窗口中选择要设置为主键的列，然后单击鼠标右键，在弹出的快捷菜单中选择“删除主键”命令，如图 4.14 所示。

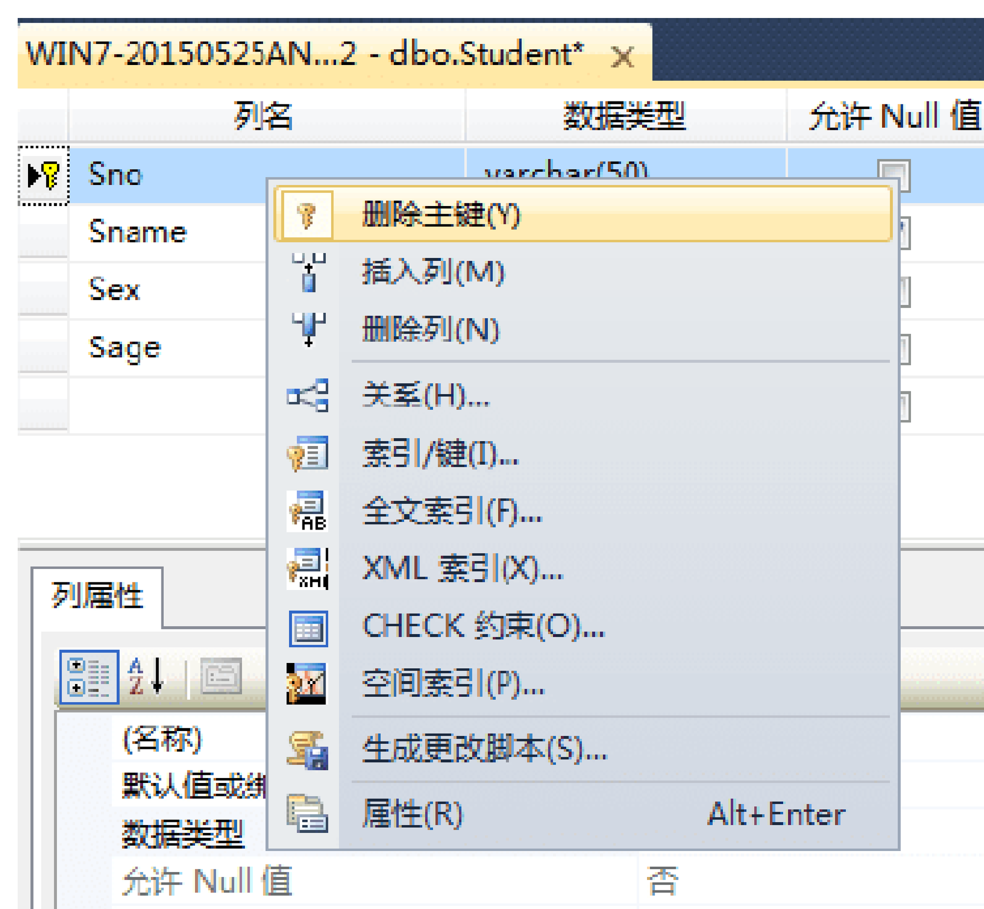


图 4.14　删除主键

使用 SQL 语句删除主键约束的语法格式如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,…n]
```

【例 4.05】　删除 tb_Student 表中的主键约束。（**实例位置：资源包\源码\04\4.05**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE tb_Student
DROP CONSTRAINT PRM_ID
```

4.5.3　唯一约束

唯一（UNIQUE）约束用于强制实施列集中值的唯一性。根据 UNIQUE 约束，表中的任何两行都不能有相同的列值。另外，主键也强制实施唯一性，但主键不允许 NULL 作为一个唯一值。

1．创建唯一约束

以界面方式创建唯一约束的操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）在“人员信息表”上单击鼠标右键，在弹出的快捷菜单中选择“设计”命令。

（4）鼠标右键单击该表中的“联系电话”这一列，在弹出的快捷菜单中选择“索引/键”命令，或者在工具栏中单击按钮，弹出“索引/键”窗体，如图 4.15 和图 4.16 所示。

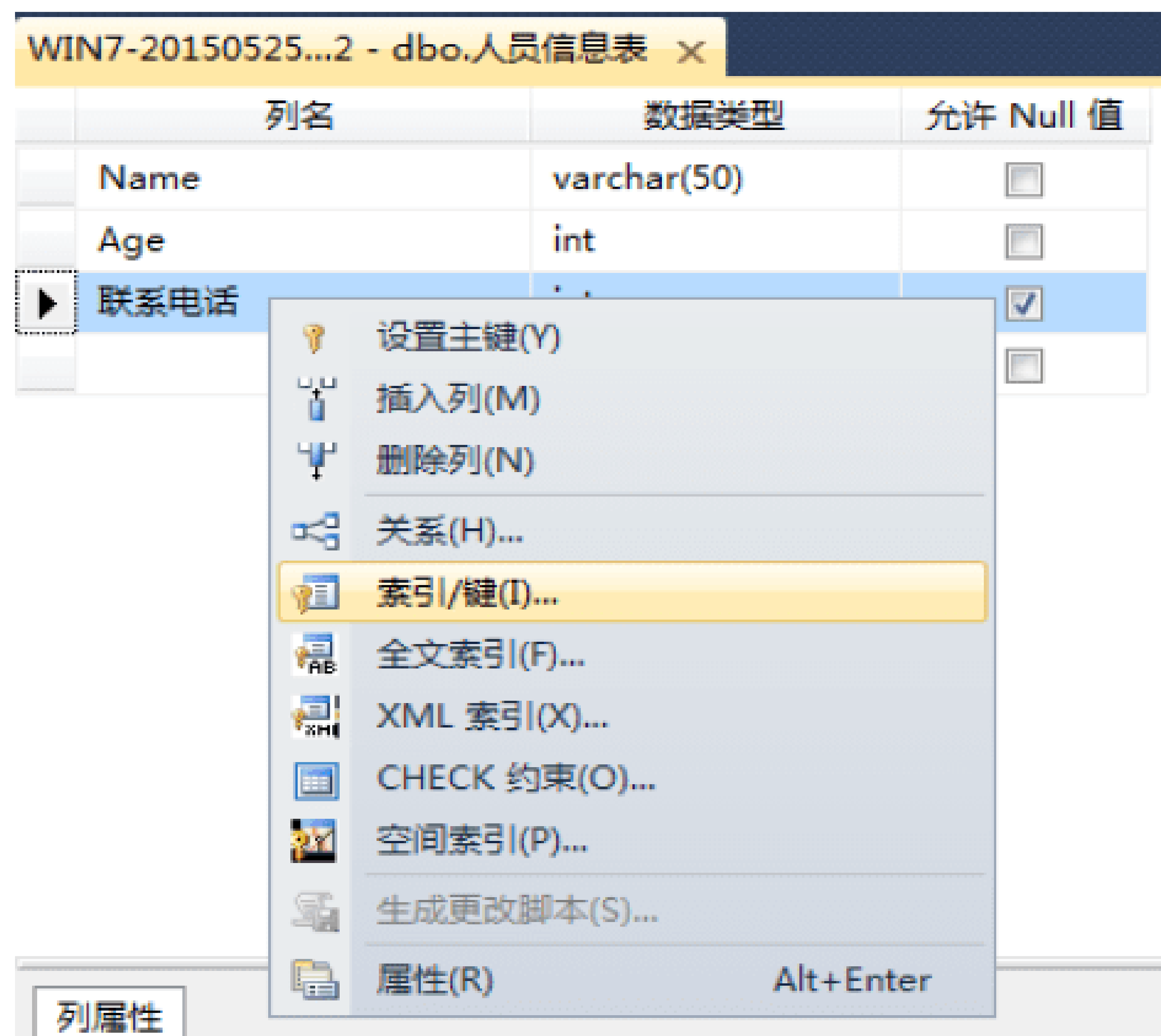


图 4.15　选择“索引/键”命令

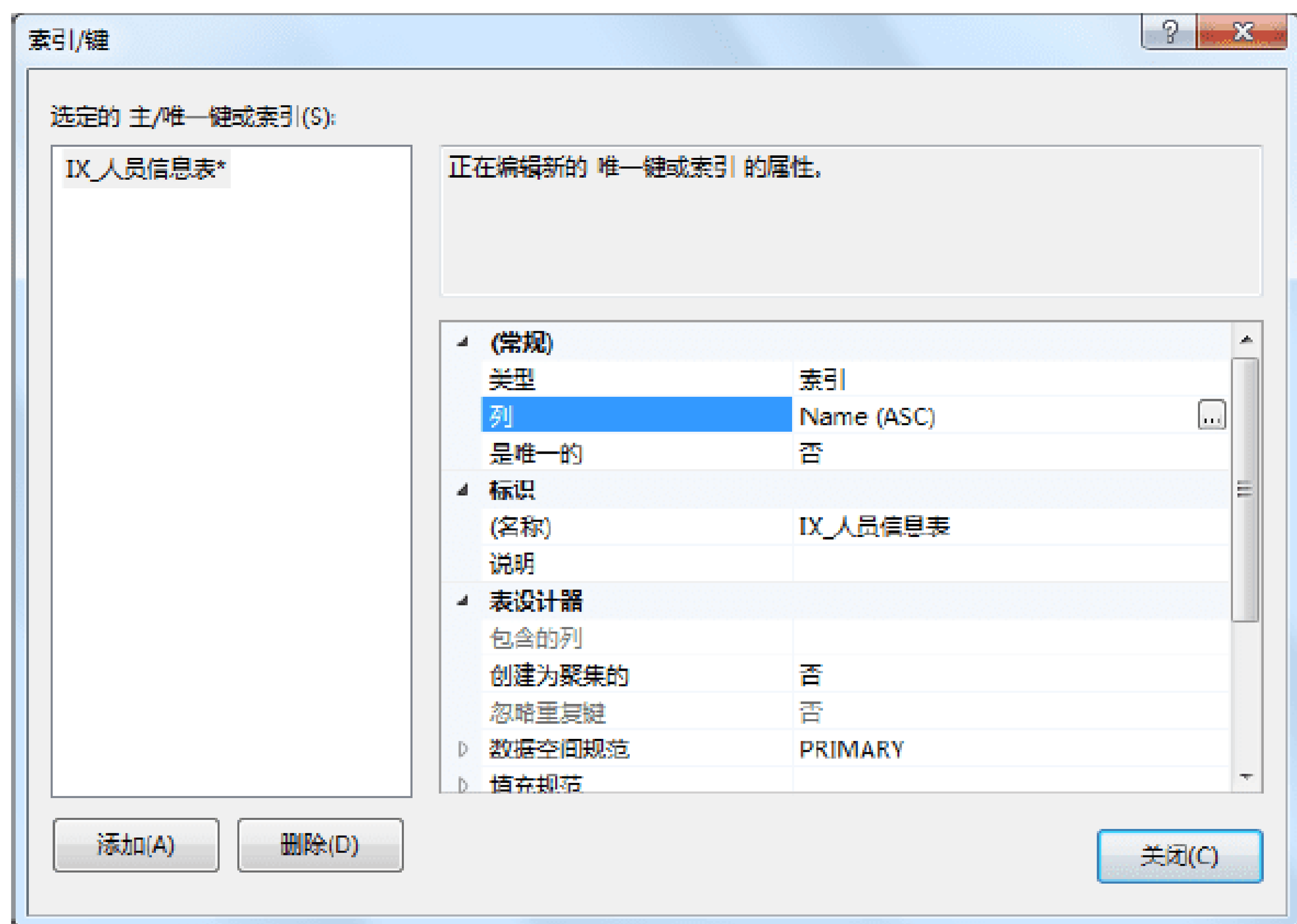


图 4.16　“索引/键”窗体

（5）在该窗体中选择“列”，并单击后面的按钮，选择要设置唯一约束的列，此处选择的是“联系电话”列，并设置该列的排列顺序。

（6）在“是唯一的”下拉列表中选择“是”，就可以将选择的列设置唯一约束。

（7）在“（名称）”文本框中输入该约束的名称，设置完成后单击“关闭”按钮即可。设置后的结果如图 4.17 所示。

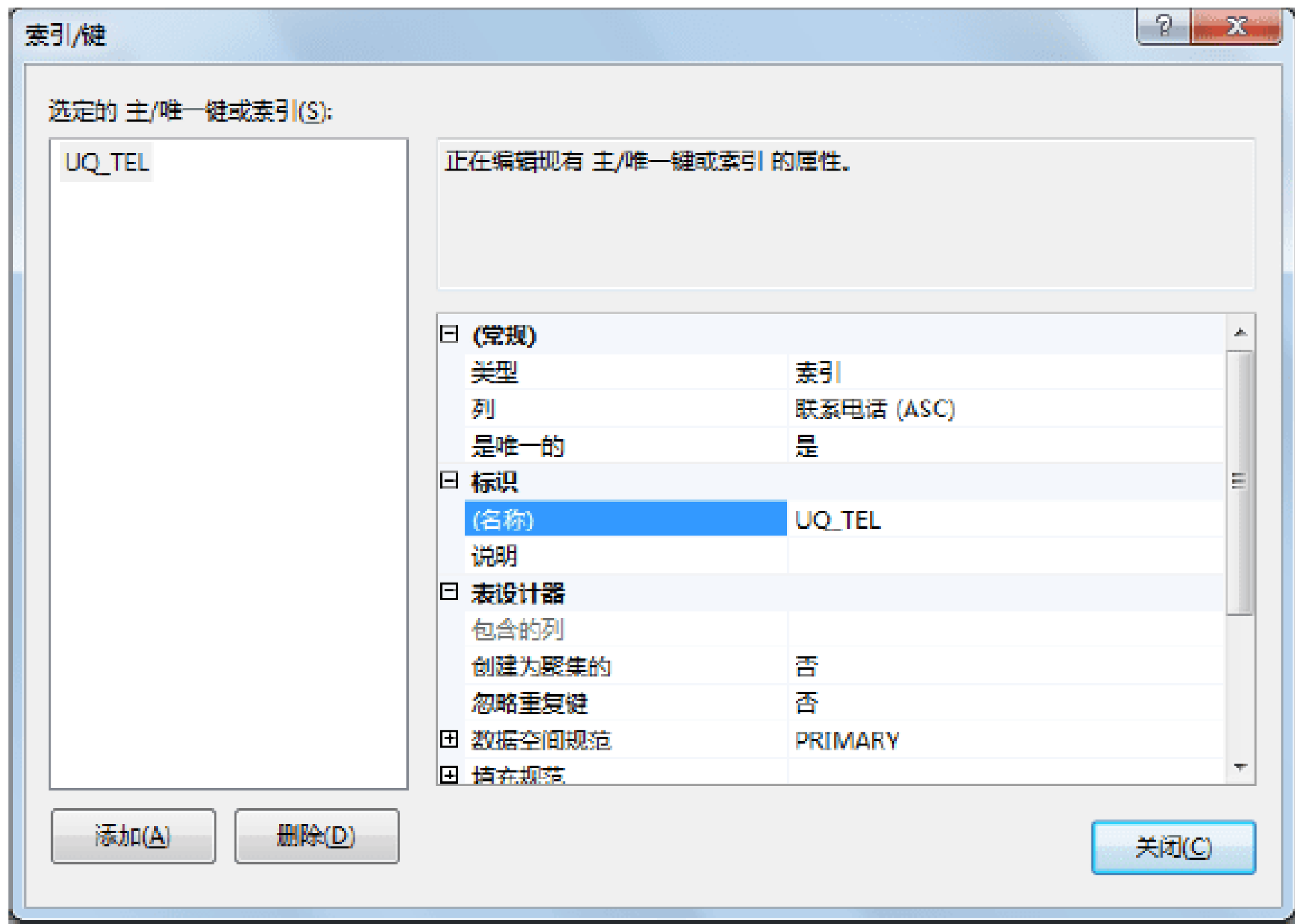


图 4.17　创建唯一约束

用 SQL 语句创建唯一约束如下。

（1）在创建表时创建唯一约束

【例 4.06】　在 db_2012 数据库中创建数据表 Employee，并将字段 ID 设置唯一约束。（**实例位置：资源包\源码\04\4.06**）

SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[Employee](
[ID] [int] CONSTRAINT UQ_ID UNIQUE,
[Name] [char](50),
[Sex] [char](2),
[Age] [int]
)
```

（2）在现有表中创建唯一约束

以 SQL 语句的方式在现有表中创建唯一约束的语法格式如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE [CLUSTERED | NONCLUSTERED]
{(column [,…n])}
```

参数说明如下。

☑　table_name：要创建唯一约束的表名称。

☑　constraint_name：唯一约束名称。

☑　column：要创建唯一约束的列名称。

【例 4.07】　将 Employee 表中的 ID 字段指定设置唯一约束。（**实例位置：资源包\源码\04\4.07**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee
ADD CONSTRAINT Unique_ID
UNIQUE(ID)
```

2．修改唯一约束

若要修改 UNIQUE 约束，必须首先删除现有的 UNIQUE 约束，然后用新定义重新创建。

3．删除唯一约束

（1）以界面的方式删除唯一约束的步骤如下。

如果想修改唯一约束，可重新设置图 4.17 中的信息，如重新选择列、重新设置唯一约束的名称等，然后单击“关闭”按钮，将该窗体关闭，最后再单击按钮，将修改后的表保存。

（2）以 SQL 语句的方式删除唯一约束的语法格式如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,...n]
```

【例 4.08】　删除 Employee 表中的唯一约束。（**实例位置：资源包\源码\04\4.08**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee
DROP CONSTRAINT Unique_ID
```

4.5.4　检查约束

检查（CHECK）约束可以强制域的完整性。CHECK 约束类似于 FOREIGN KEY 约束，可以控制放入列中的值。但是，它们在确定有效值的方式上有所不同：FOREIGN KEY 约束从其他表获得有效值列表，而 CHECK 约束通过不基于其他列中的数据的逻辑表达式确定有效值。

1．创建检查约束

以界面的方式创建检查约束的操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）鼠标右键单击要创建约束的表，在弹出的快捷菜单中选择“设计”命令。

（4）鼠标右键单击该表中的某一列，在弹出的快捷菜单中选择“CHECK 约束”命令，如图 4.18 所示。在弹出的窗体中设置约束的表达式，例如，输入 sex='女' OR sex='男'，表示性别只能是女或男，如图 4.19 所示。

用 SQL 语句创建检查约束如下。

（1）在创建表时创建检查约束

【例 4.09】　创建数据表 Employee，并将字段 Sex 设置检查约束，在输入性别字段时，只能接受“男”或者“女”，而不能接受其他数据。（**实例位置：资源包\源码\04\4.09**）

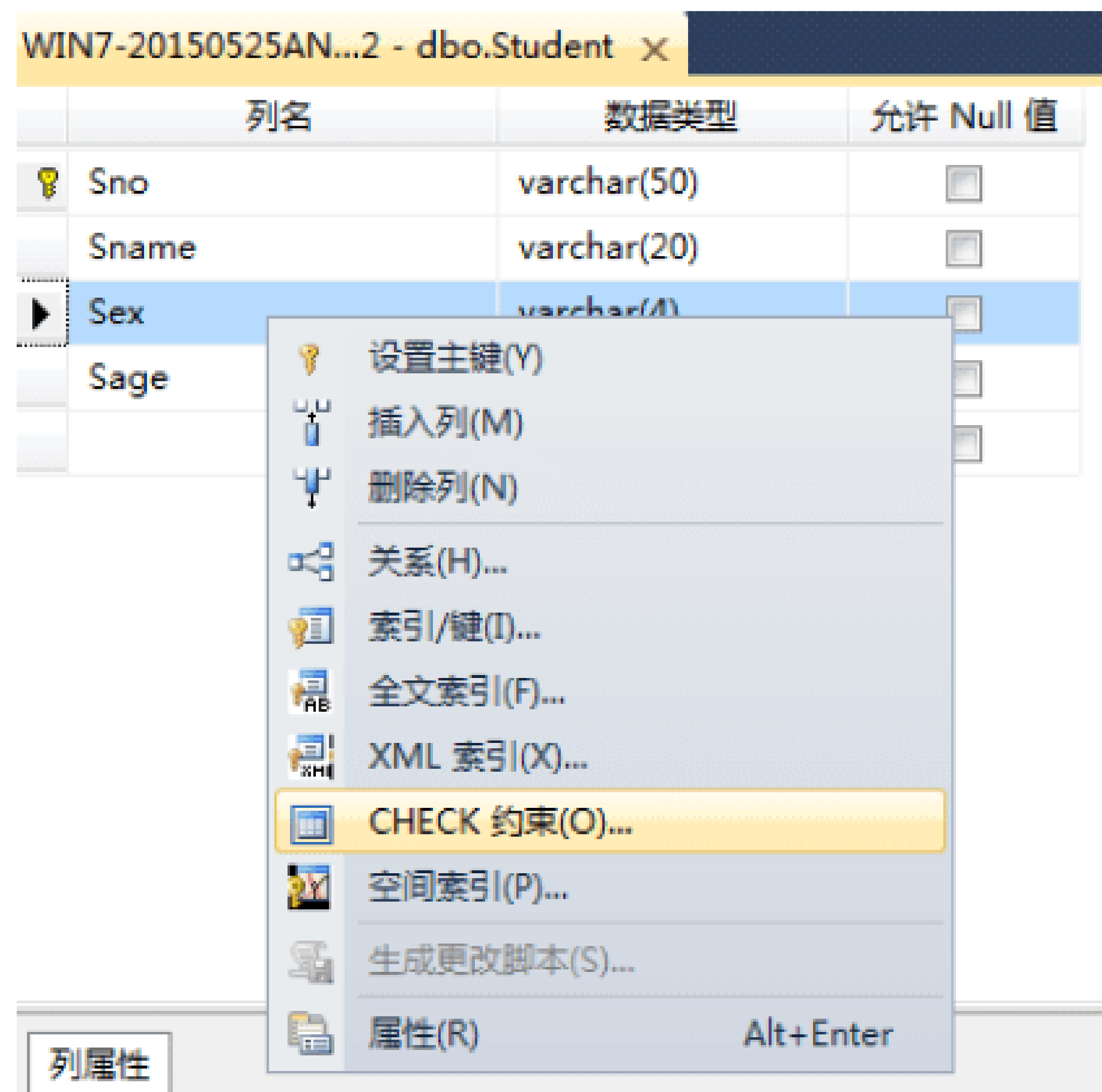


图 4.18　选择“CHECK 约束”命令

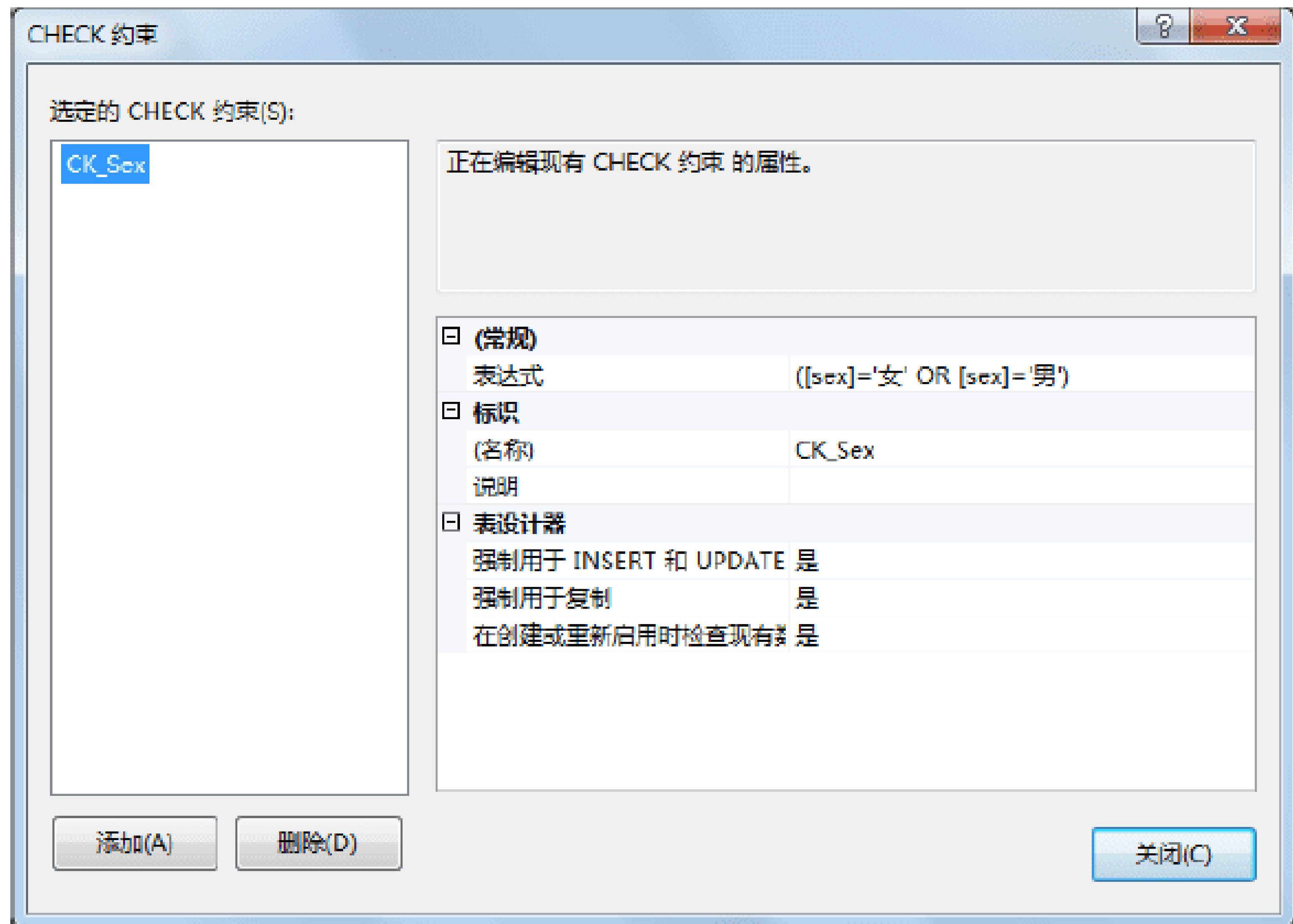


图 4.19　创建“CHECK 约束”

SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[Employee](
[ID] [int],
[Name] [char](50),
[Sex] [char](2) CONSTRAINT CK_Sex Check(sex in('男','女')),
[Age] [int]
)
```

（2）在现有表中创建检查约束

以 SQL 语句方式在现有表中创建检查约束的语法格式如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
CHECK (logical_expression)
```

参数说明如下。

☑　table_name：要创建检查约束的表名称。

☑　constraint_name：检查约束名称。

☑　logical_expression：要检查约束的条件表达式。

【例 4.10】　为 Employee 表中的 Sex 字段设置检查约束，在输入性别的时候只能接受“女”，不能接受其他字段。（**实例位置：资源包\源码\04\4.10**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE [Employee]
ADD CONSTRAINT Check_Sex Check(sex='女')
```

2．修改检查约束

修改表中某列的 CHECK 约束使用的表达式，必须首先删除现有的 CHECK 约束，然后使用新定

义重新创建，才能修改 CHECK 约束。

3．删除检查约束

（1）以界面的方式删除检查约束

如果想将创建的检查约束删除，单击图 4.19 中的“删除”按钮，就可以将创建的检查约束删除，然后单击“关闭”按钮，将该窗体关闭，最后再单击按钮，将修改后的表保存。

（2）以 SQL 语句的方式删除检查约束

删除检查约束的语法格式如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,…n]
```

删除 Employee 表中的检查约束，SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee
DROP CONSTRAINT Check_Sex
```

4.5.5　默认约束

在创建或修改表时可通过定义默认（DEFAULT）约束来创建默认值。默认值可以是计算结果为常量的任何值，如常量、内置函数或数学表达式。这将为每一列分配一个常量表达式作为默认值。

1．创建默认约束

以界面的方式创建默认约束的操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）在 Student 表上单击鼠标右键，在弹出的快捷菜单中选择“设计”命令。

（4）选择该表中 Sex 这一列，在下面的列属性中选择“默认值或绑定”，在其后面的文本框中输入要设置约束的值，例如，输入“'男'”，表示该列的默认性别为男，如图 4.20 所示。

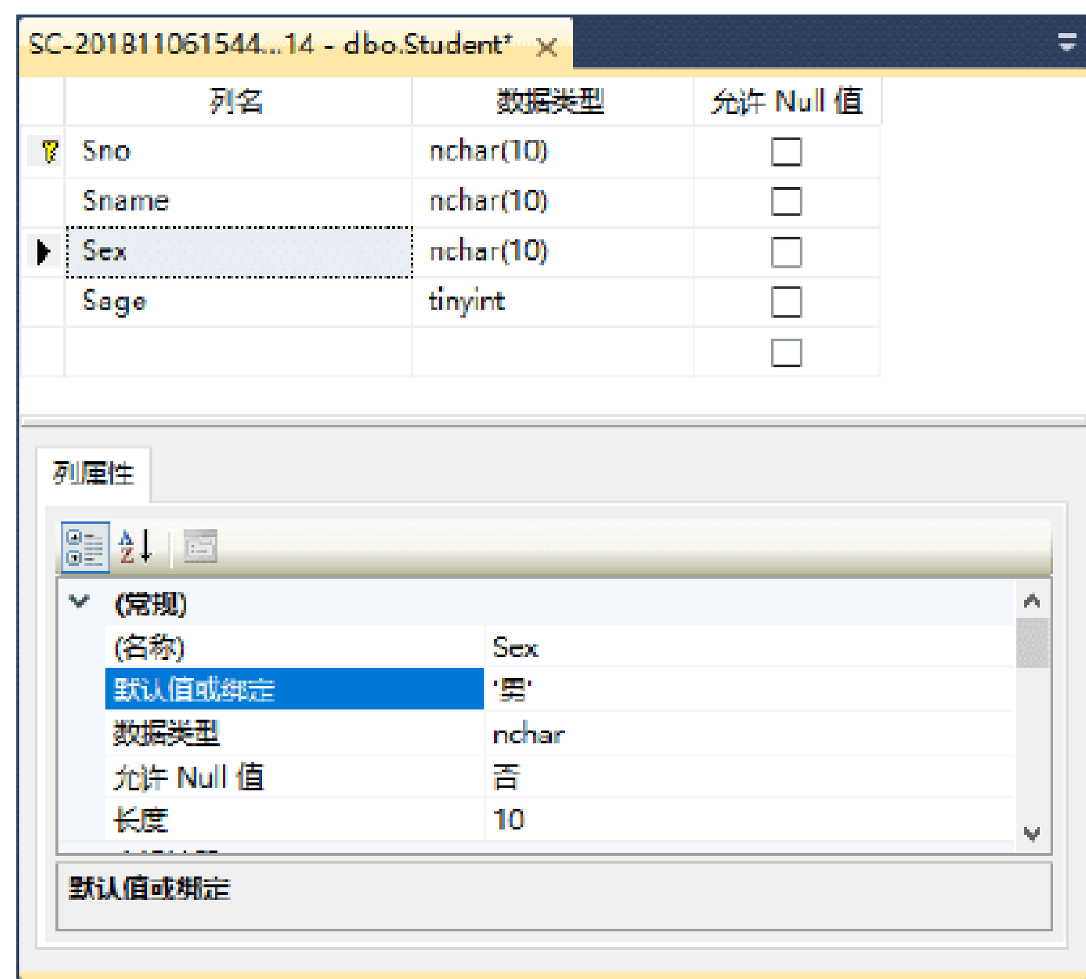


图 4.20　创建默认约束

（5）最后单击按钮，就可以将设置完默认约束的表保存。

用 SQL 语句创建默认约束如下。

（1）在创建表时创建默认约束

【例 4.11】 创建数据表 Employee，并为字段 Sex 设置默认约束“女”。（**实例位置：资源包\源码\04\4.11**）

SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[Employee](
[ID] [int],
[Name] [char](50) ,
[Sex] [char](2) CONSTRAINT Def_Sex Default '女',
[Age] [int]
)
```

（2）在现有表中创建默认约束

以 SQL 语句的方式在现有表中创建默认约束的语法格式如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
DEFAULT constant_expression [FOR column_name]
```

参数说明如下。

- ☑ table_name：要创建默认约束的表名称。
- ☑ constraint_name：默认约束名称。
- ☑ constant_expression：默认值。

【例 4.12】 为 Employee 表中的 Sex 字段设置默认约束“男”。（**实例位置：资源包\源码\04\4.12**）

SQL 语句如下：

```
ALTER TABLE [Employee]
ADD CONSTRAINT Default_Sex
DEFAULT '男' FOR Sex
```

2．修改默认约束

修改表中某列的默认约束使用的表达式，必须首先删除现有的默认约束，然后使用新定义重新创建，才能修改默认约束。

3．删除默认约束

以界面的方式删除默认约束的方法如下：如果想删除默认约束，将“列属性”中的“默认值或绑定”文本框中的内容清空即可，最后再单击按钮，将修改后的表保存。

以 SQL 语句的方式删除默认约束的语法格式如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,…n]
```

【例 4.13】　删除 Employee 表中的默认约束。（**实例位置：资源包\源码\04\4.13**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee
DROP CONSTRAINT Default_Sex
```

4.5.6　外键约束

通过定义外键（FOREIGN KEY）约束来创建外键。在外键引用中，当一个表的列被引用作为另一个表的主键值的列时，就在两表之间创建了链接。这个列就成为第二个表的外键。

1．创建外键约束

以界面的方式创建外键约束的操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）在 EMP 表上单击鼠标右键，在弹出的快捷菜单中选择“设计”命令。

（4）鼠标右键单击该表中的某一列，在弹出的快捷菜单中选择“关系”，或者在工具栏中单击按钮，弹出“外键关系”窗体，单击该窗体中的“添加”按钮，添加要选中的关系，如图 4.21 所示。

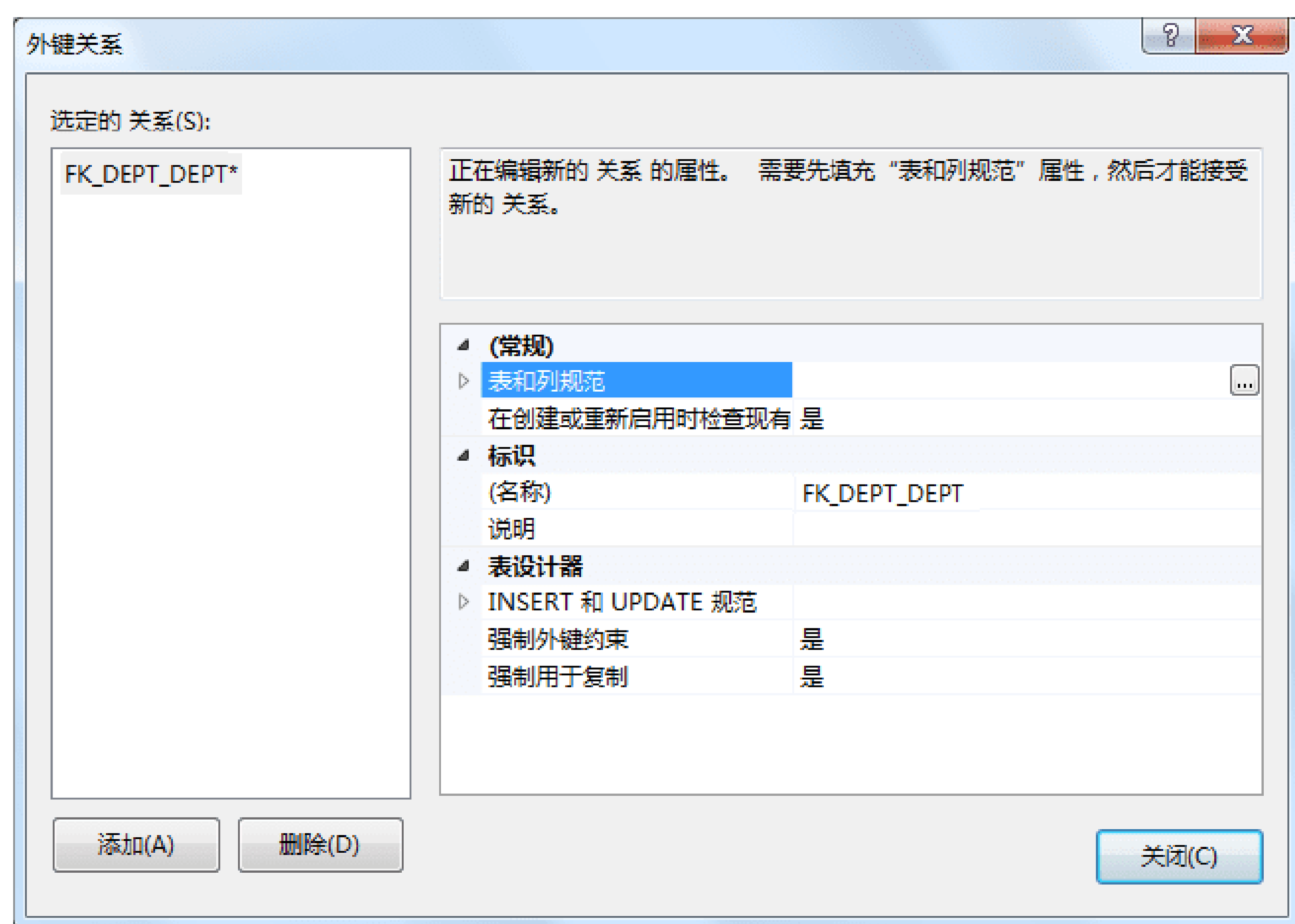


图 4.21　“外键关系”窗体

（5）在外键关系窗体中，单击“表和列规范”后面的按钮，在弹出的窗体中选择要创建外键约束的主键表和外键表，如图 4.22 所示。

（6）在“表和列”窗体中，设置关系的名称，然后选择外键要参照的主键表及使用的字段。最后单击“确定”按钮，回到“外键关系”窗体中，如图 4.23 所示。

（7）单击“关闭”按钮，将该窗体关闭，最后再单击按钮，将设置约束后的表保存。

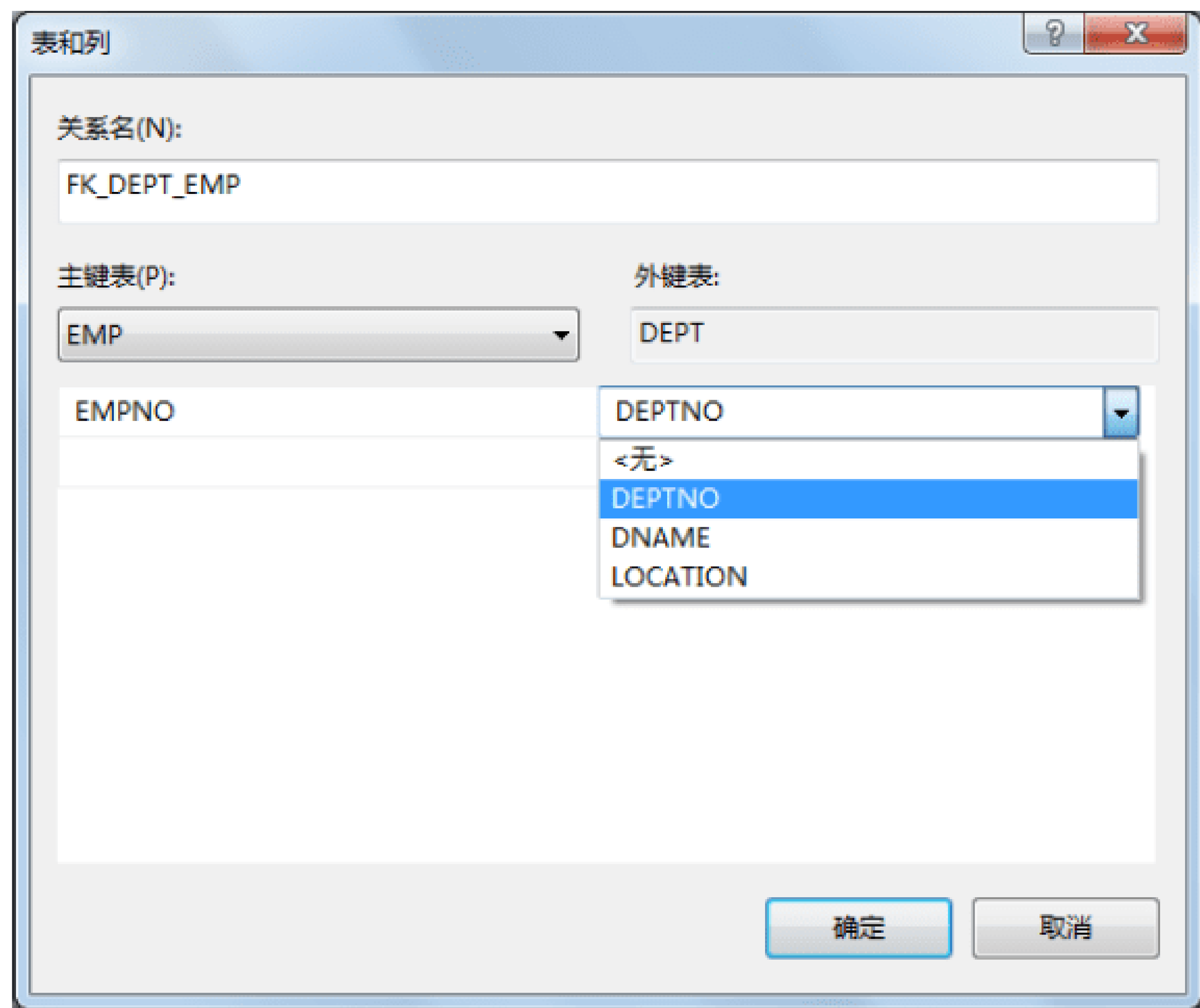


图 4.22 “表和列”窗体

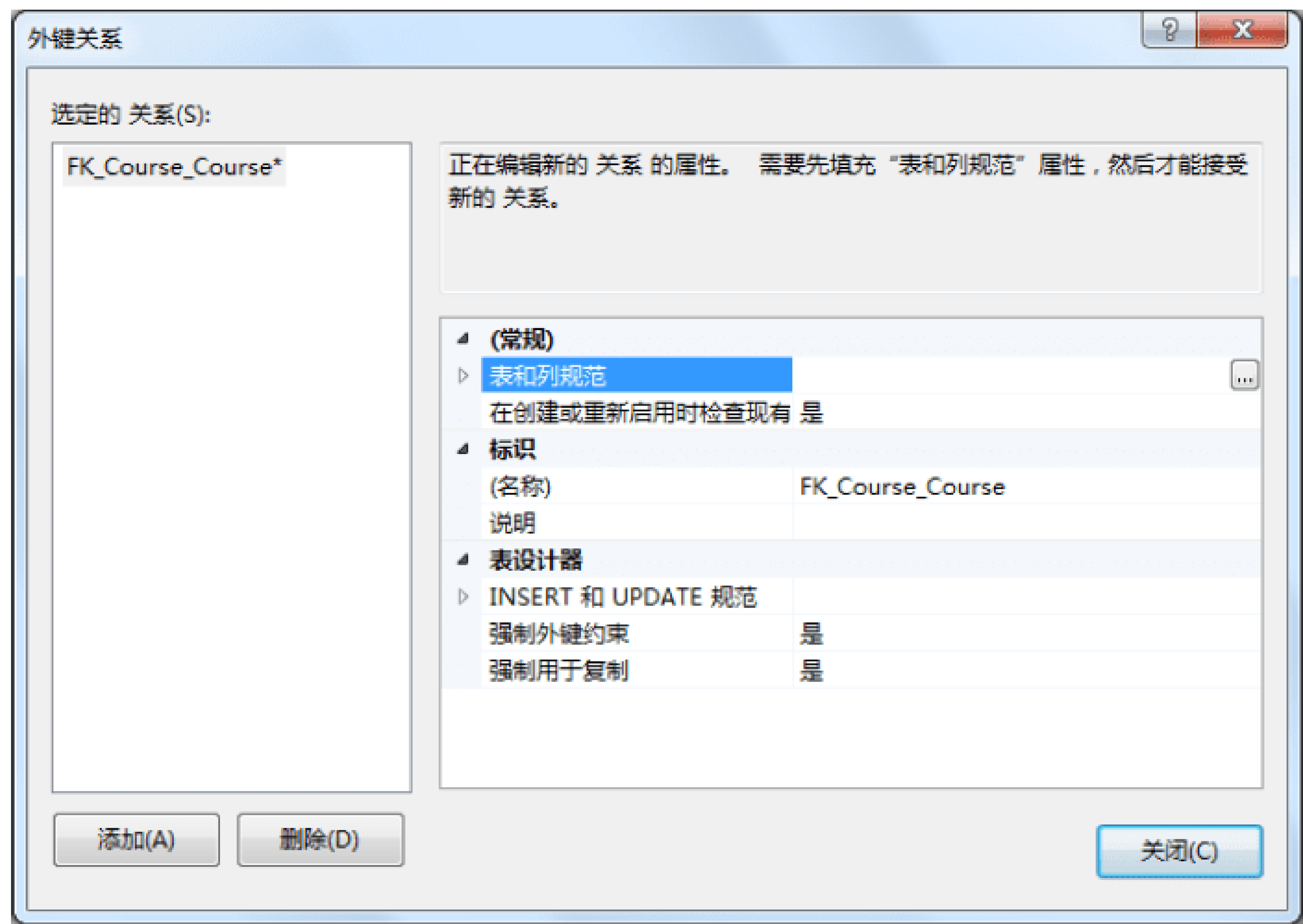


图 4.23 “外键关系”窗体

用 SQL 语句创建外键约束如下。

（1）在创建表时创建外键约束

【例 4.14】 创建表 Laborage，并为 Laborage 表创建外键约束，该约束把 Laborage 中的编号（ID）字段和表 Employee 中的编号（ID）字段关联起来，实现 Laboratory 中的编号（ID）字段的取值要参照表 Employee 中编号（ID）字段的数据值。（**实例位置：资源包\源码\04\4.14**）

SQL 语句如下：

```
USE db_2012
CREATE TABLE Laborage
(
```

```
ID INT,
Wage MONEY,
CONSTRAINT FKEY_ID
FOREIGN KEY (ID)
REFERENCES Employee(ID)
)
```



说明

FOREIGN KEY (ID)中的 ID 字段为 Laborage 表中的编号（ID）字段。

（2）在现有表中创建外键约束

用 SQL 语句的方式在现有表中创建外键约束的语法格式如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
[FOREIGN KEY]{(column_name[,…n])}
REFERENCES ref_table[(ref_column_name[,…n])]
```

创建外键约束语句的参数及说明如表 4.3 所示。

表 4.3　创建外键约束语句的参数及说明

参　数	描　述
table_name	要创建外键的表名称
constraint_name	外键约束名称
FOREIGN KEY...REFERENCES	为列中的数据提供引用完整性的约束。FOREIGN KEY 约束要求列中的每个值在被引用表中对应的被引用列中都存在。FOREIGN KEY 约束只能引用被引用表中为 PRIMARY KEY 或 UNIQUE 约束的列或被引用表中在 UNIQUE INDEX 内引用的列
ref_table	FOREIGN KEY 约束所引用的表名
(ref_column[,...n])	FOREIGN KEY 约束所引用的表中的一列或多列

【例 4.15】　将 Employee 表中的 ID 字段设置为 Laborage 表中的外键。（**实例位置：资源包\源码\04\4.15**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Laborage
ADD CONSTRAINT Fkey_ID
FOREIGN KEY (ID)
REFERENCES Employee(ID)
```

2．修改外键约束

修改表中某列的外键约束。必须首先删除现有的外键约束，然后使用新定义重新创建，才能修改外键约束。

3．删除默认约束

如果想修改外键约束，可重新设置图 4.23 中的信息，如重新选择外键要参照的主键表及使用的字段、重新设置外键约束的名称等，然后单击“关闭”按钮，将该窗体关闭，最后再单击按钮，将修改后的表保存。或者使用 SQL 语句删除外键约束。

删除外键约束的语法格式如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,...n]
```

【例 4.16】 删除 Employee 表中的默认约束。（**实例位置：资源包\源码\04\4.16**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Laborage
DROP CONSTRAINT FKEY_ID
```


视频讲解

4.6 修 改 表

使用 ALTER TABLE 语句可以修改表的结构，语法格式如下：

```
ALTER TABLE [database_name . [schema_name] . | schema_name .] table_name
{
ALTER COLUMN column_name
{
[type_schema_name.] type_name [({precision [, scale]
| max | xml_schema_collection})]
[COLLATE collation_name]
[NULL | NOT NULL]
| {ADD | DROP}
{ROWGUIDCOL | PERSISTED| NOT FOR REPLICATION | SPARSE }
}
| [WITH {CHECK | NOCHECK}]
| ADD
{
<column_definition>
| <computed_column_definition>
| <table_constraint>
| <column_set_definition>
} [,...n]
| DROP
{
[CONSTRAINT] constraint_name
[WITH (<drop_clustered_constraint_option> [,...n])]
| COLUMN column_name
} [,...n]
```

ALTER TABLE 语句的参数及说明如表 4.4 所示。

表 4.4　ALTER TABLE 语句的参数及说明

参　　数	描　　述
database_name	创建表时所在的数据库的名称
schema_name	表所属架构的名称
table_name	要更改的表的名称
ALTER COLUMN	指定要更改命名列
column_name	要更改、添加或删除的列的名称
[type_schema_name.] type_name	更改后的列的新数据类型或添加的列的数据类型
precision	指定的数据类型的精度
scale	指定的数据类型的小数位数
max	仅应用于 VARCHAR、NVARCHAR 和 VARBINARY 数据类型
xml_schema_collection	仅应用于 XML 数据类型
COLLATE <collation_name>	指定更改后的列的新排序规则
NULL | NOT NULL	指定列是否可接受空值
[{ADD | DROP} ROWGUIDCOL]	指定在指定列中添加或删除 ROWGUIDCOL 属性
[{ADD | DROP} PERSISTED]	指定在指定列中添加或删除 PERSISTED 属性
DROP NOT FOR REPLICATION	指定当复制代理执行插入操作时，标识列中的值将增加
SPARSE	指示列为稀疏列。稀疏列已针对 NULL 值进行了存储优化。不能将稀疏列指定为 NOT NULL
WITH CHECK | WITH NOCHECK	指定表中的数据是否用新添加的或重新启用的 FOREIGN KEY 或 CHECK 约束进行验证
ADD	指定添加一个或多个列定义、计算列定义或者表约束
DROP {[CONSTRAINT] constraint_name | COLUMN column_name}	指定从表中删除 constraint_name 或 column_name。可以列出多个列或约束
WITH <drop_clustered_constraint_option>	指定设置一个或多个删除聚集约束选项

【例 4.17】 向 db_2012 数据库中的 tb_Student 表中添加 Sex 字段。（**实例位置：资源包\源码\04\4.17**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE tb_Student
ADD Sex char(2)
```

【例 4.18】 删除 db_2012 数据库中 tb_Student 中的 Sex 字段。（**实例位置：资源包\源码\04\4.18**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE tb_Student
DROP COLUMN Sex
```


视频讲解

4.7 删 除 表

使用 DROP TABLE 语句可以删除数据表，其语法格式如下：

```
DROP TABLE [database_name . [schema_name] . | schema_name .]
          table_name [,...n] [;]
```

参数说明如下。

☑ database_name：要在其中删除表的数据库的名称。

☑ schema_name：表所属架构的名称。

☑ table_name：要删除的表的名称。

【例 4.19】 删除 db_2012 数据库中 tb_Student 表。（**实例位置：资源包\源码\04\4.19**）

SQL 语句如下：

```
USE db_2012
DROP TABLE tb_Student
```

4.8 小 结

本章介绍了数据表的基础知识，数据表的创建、修改和删除以及表中的约束。读者可以使用 SQL Server 2014 界面方式完成创建和管理数据表的工作。

第 5 章

操作表数据

（视频讲解：24 分钟）

本章主要介绍分区表，操作表数据和表与表的关联。通过本章的学习，读者可以熟悉分区表的创建，并能够掌握操作表数据的方法。

学习摘要：

- 分区表的创建
- 添加表记录
- 修改表记录
- 删除表记录
- 表与表之间的关联


视频讲解

5.1 分 区 表

5.1.1 分区表概述

分区表是把数据库按照某种标准划分成区域存储在不同的文件组中，使用分区可以快速有效地管理和访问数据子集，从而使大型表或索引更易于管理。合理地使用分区会很大程度上提高数据库的性能。已分区表和已分区索引的数据划分为分布于一个数据库中多个文件组的单元。数据是按水平方式分区的，因此多组行映射到单个的分区。已分区表和已分区索引支持与设计和查询标准表和索引相关的所有属性和功能，包括约束、默认值、标识和时间戳值以及触发器。因为分区表的本质是把符合不同标准的数据子集存储在一个数据库的一个或多个文件组中，通过元数据来表述数据存储逻辑地址。

决定是否实现分区主要取决于表当前的大小或将来的大小、如何使用表以及对表执行用户查询和维护操作的完善程度。通常，如果某个大型表同时满足下面的两个条件，则可能适用于进行分区。

（1）该表包含（或将包含）以多种不同方式使用的大量数据。

（2）不能按预期对表执行查询或更新，或维护开销超过了预定义的维护期。

5.1.2 界面创建分区表

以界面的方式创建分区表的步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）在 db_2012 数据库上，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，如图 5.1 所示。这时在弹出的“数据库属性”窗口的“选择页”中，选择“文件”选项，然后单击“添加”按钮，添加逻辑名称，如 Group1、Goup2、Group3、Group4，添加完之后，单击“确定”按钮，如图 5.2 所示。

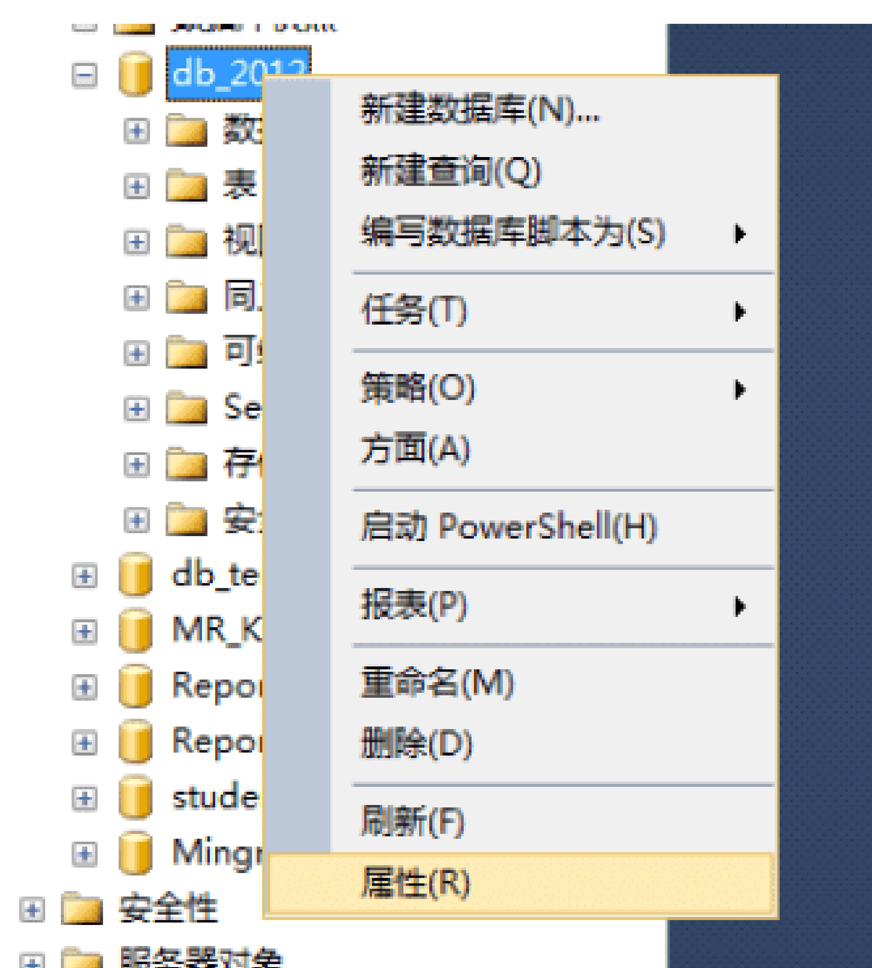


图 5.1　创建文件和文件组

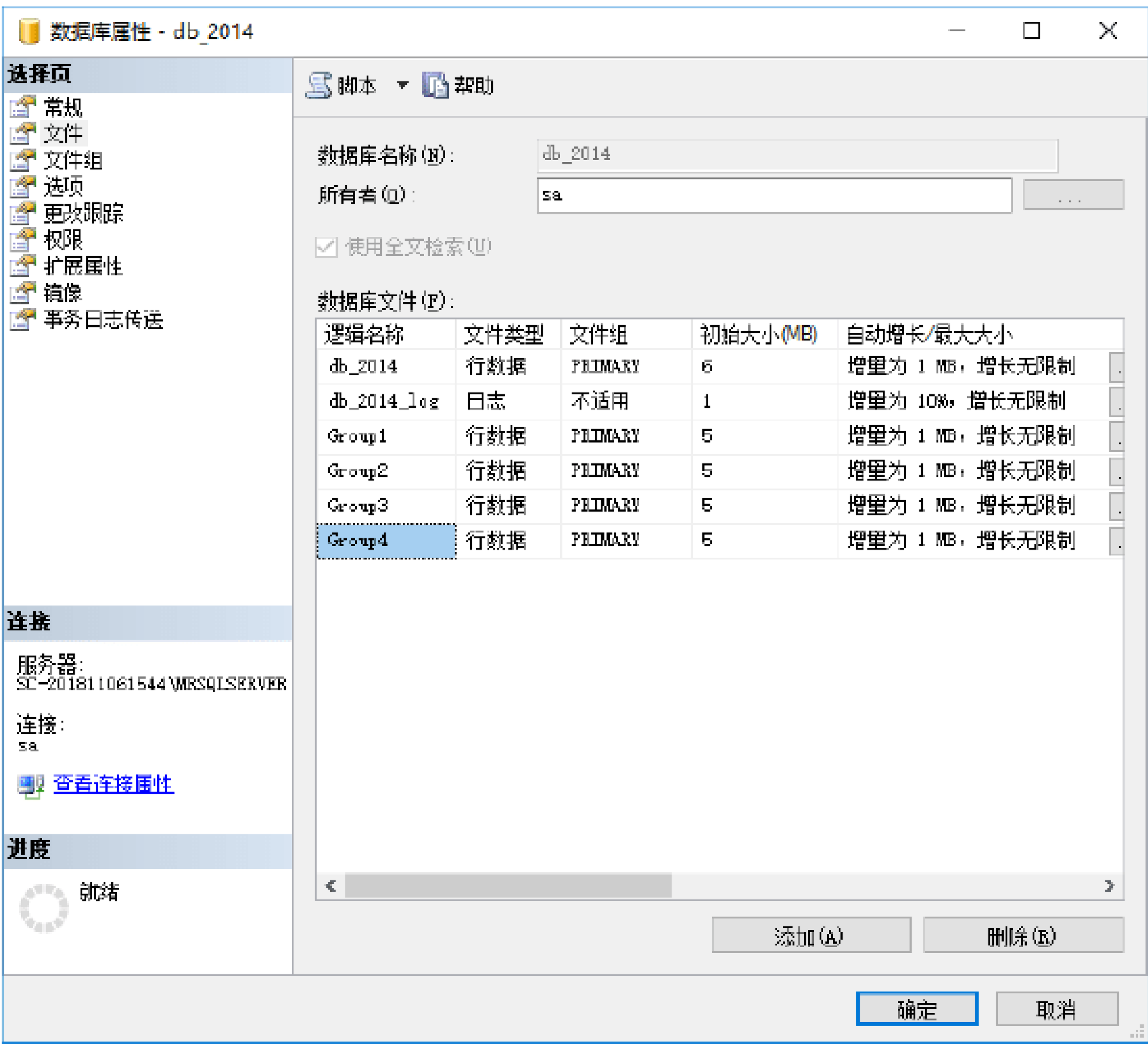


图 5.2 添加文件

（4）选择“文件组”选项，然后单击“添加”按钮，分别添加步骤（3）中的 4 个文件，然后选中在“只读”下面的复选框，最后单击“确定”按钮，如图 5.3 所示。

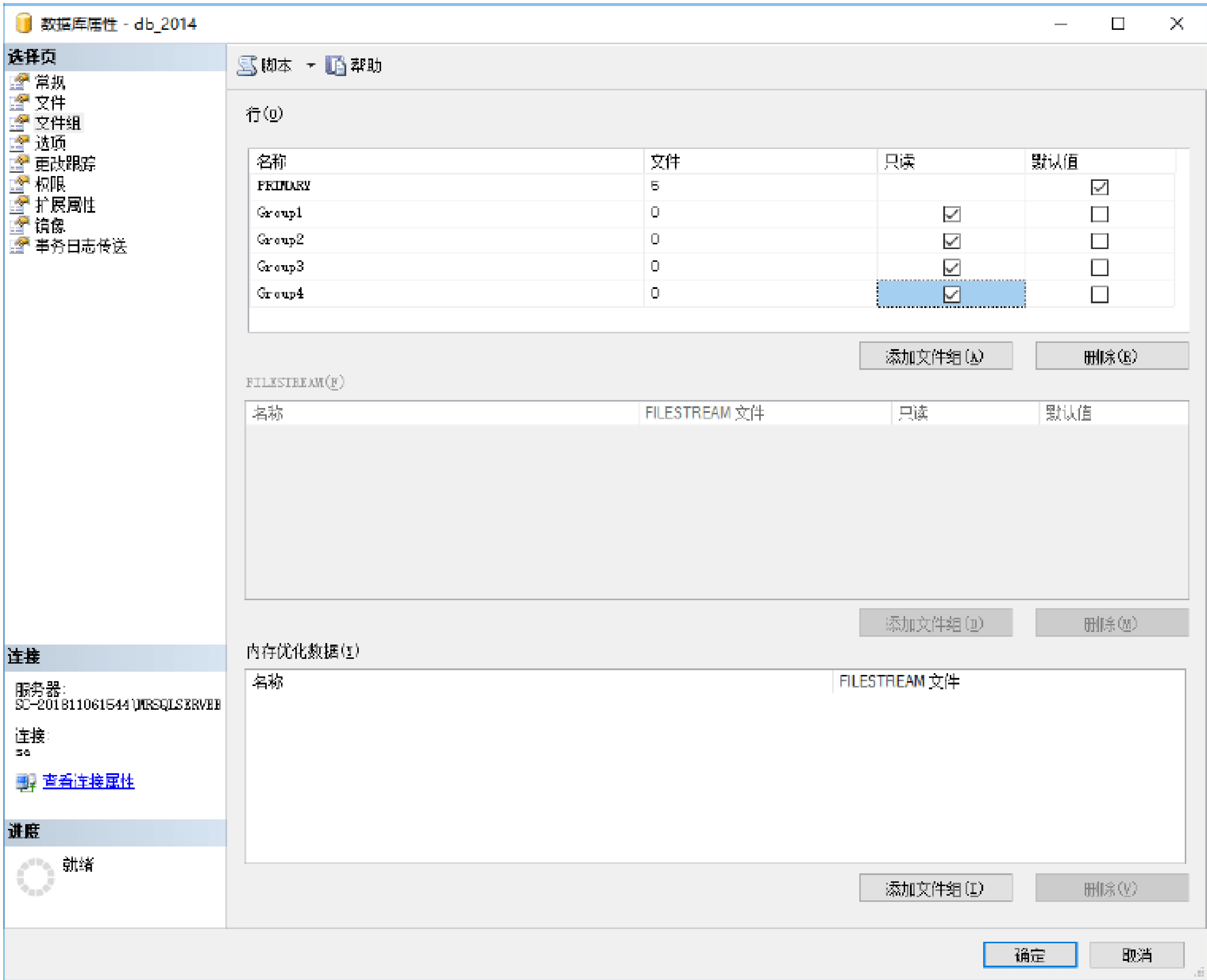


图 5.3 添加文件组

（5）在 Employee 表上单击鼠标右键，在弹出的快捷菜单中选择“存储”→“创建分区”命令，如图 5.4 所示，进入“创建分区向导”窗口，如图 5.5 所示，单击“下一步”按钮，进入“选择分区列”

界面，界面中将显示可用的分区列，选择 Age 列，如图 5.6 所示。



图 5.4　创建分区

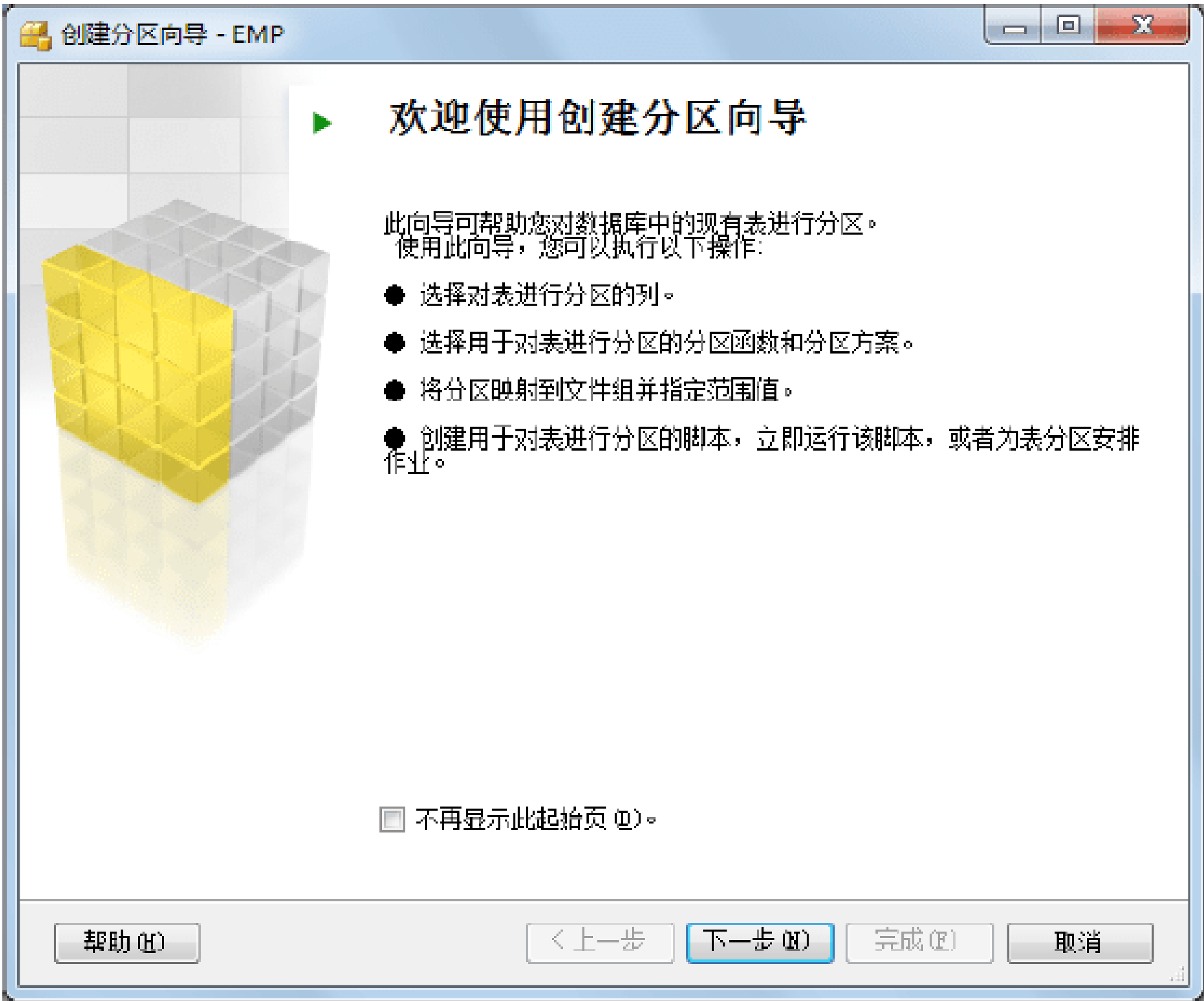


图 5.5　创建分区向导

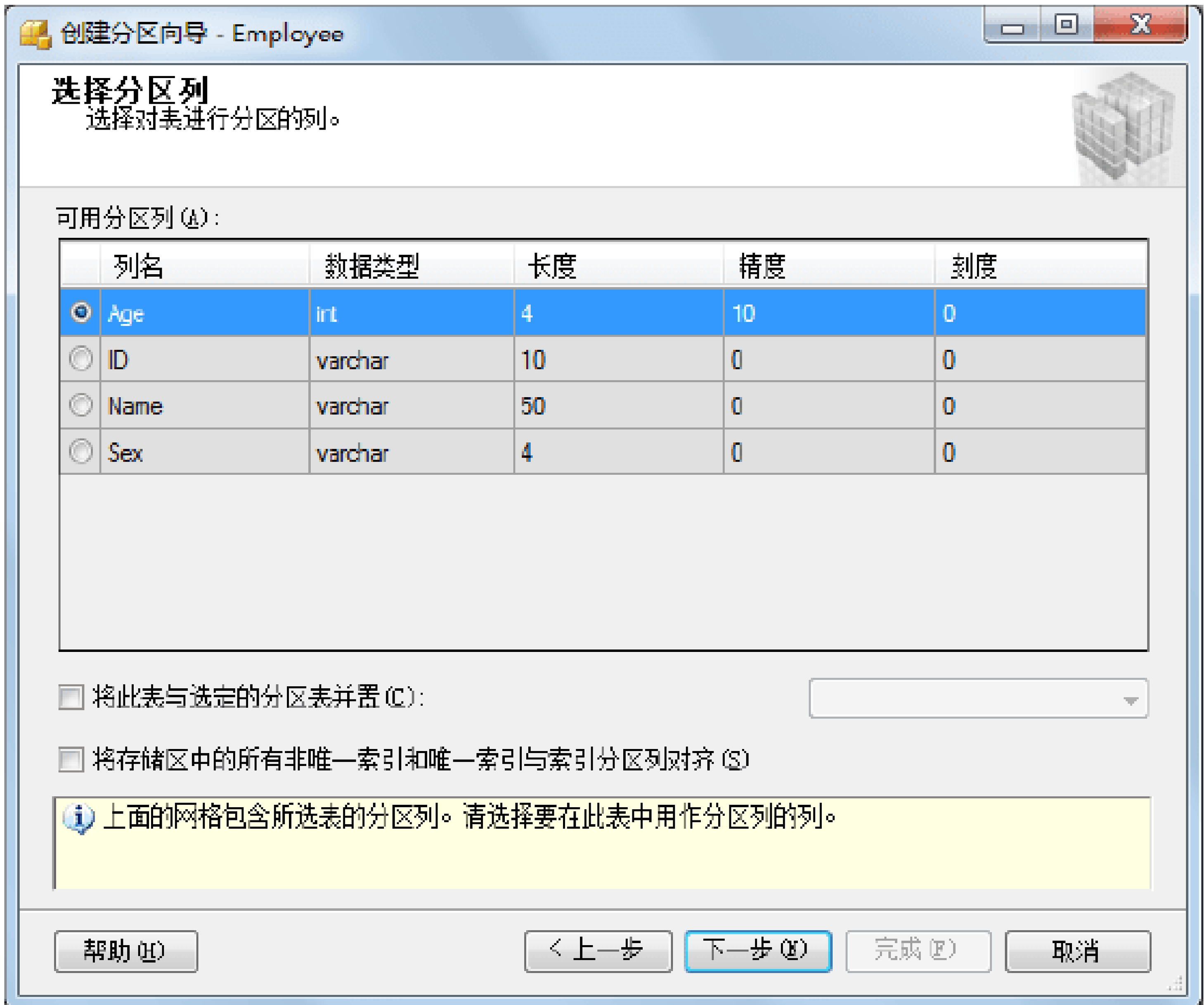


图 5.6　选择分区列

（6）单击“下一步”按钮，进入“选择分区函数”界面。在“选择分区函数”中选中“新建分区函数”单选按钮，然后在“新建分区函数”后面的文本框中输入新建分区函数的名称，如 AgeOrderFunction，如图 5.7 所示，单击“下一步”按钮。

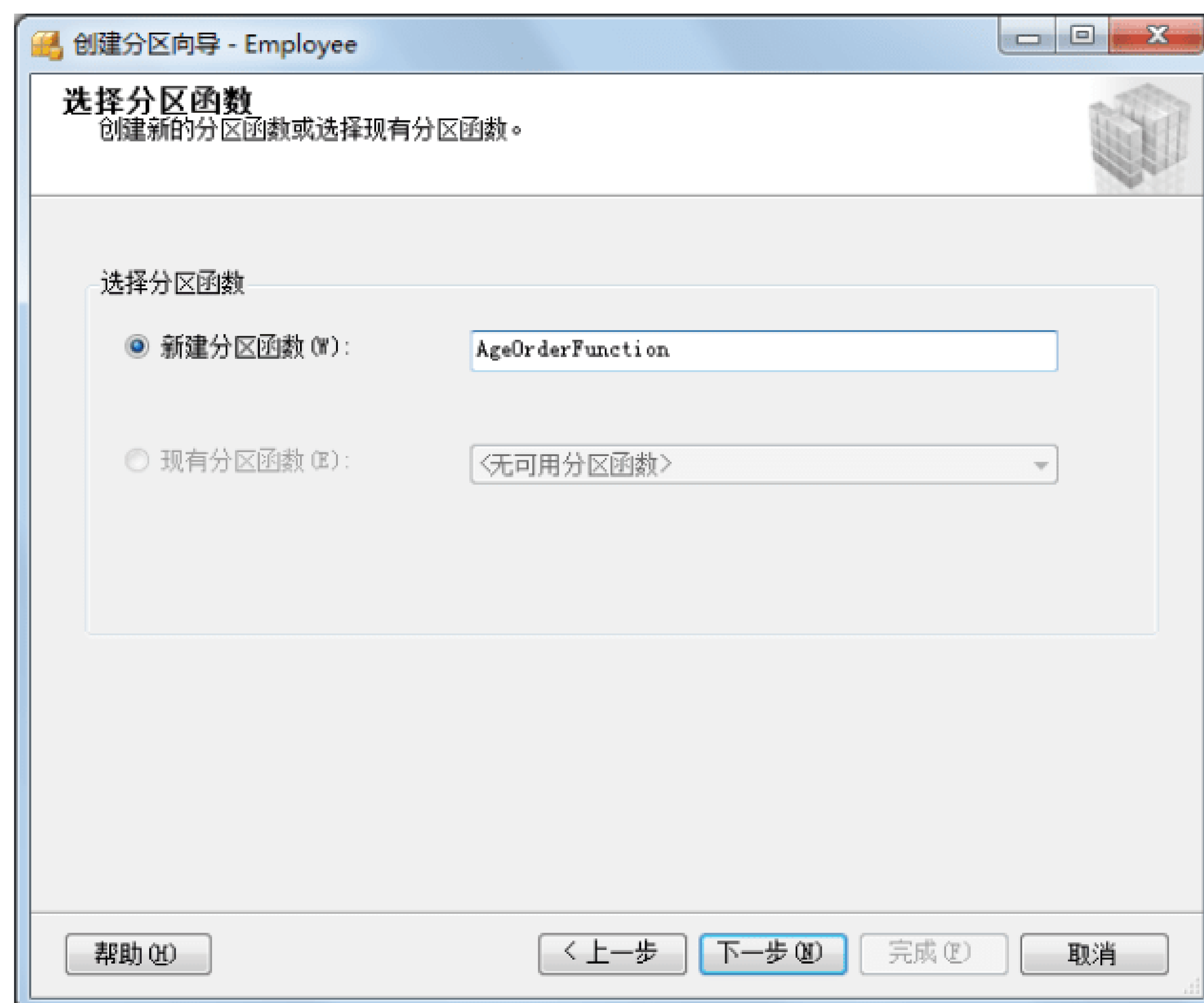


图 5.7　选择分区函数

（7）弹出“选择分区方案”界面，选中“新建分区方案”单选按钮，在“新建分区方案”后面的文本框中输入新建分区方案的名称，如 AgeOrder，如图 5.8 所示，单击“下一步”按钮。

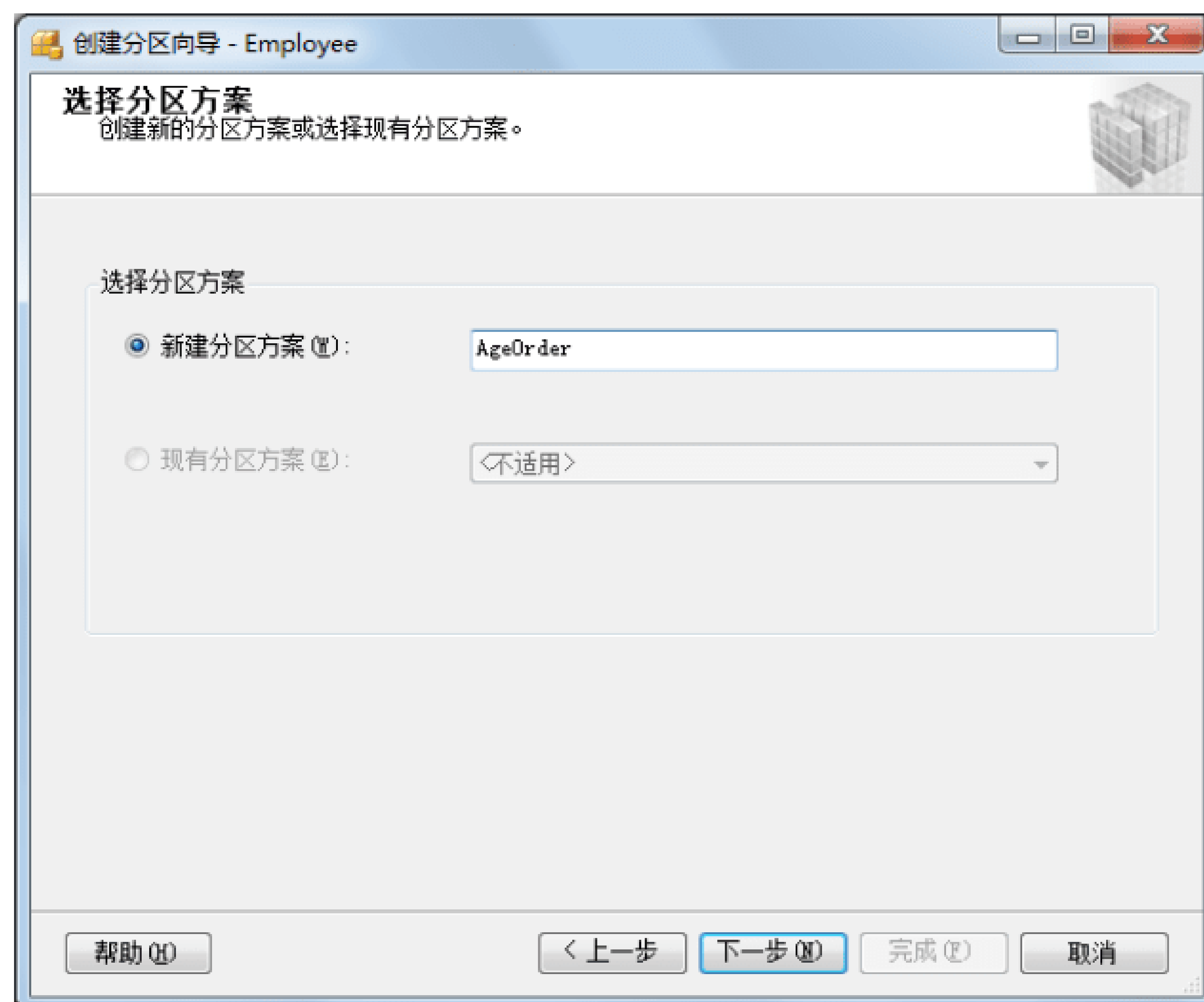


图 5.8　选择分区方案

（8）弹出“映射分区”界面，选中“左边界”单选按钮，然后选择各个分区要映射到的文件组，

如图 5.9 所示，单击“下一步”按钮。

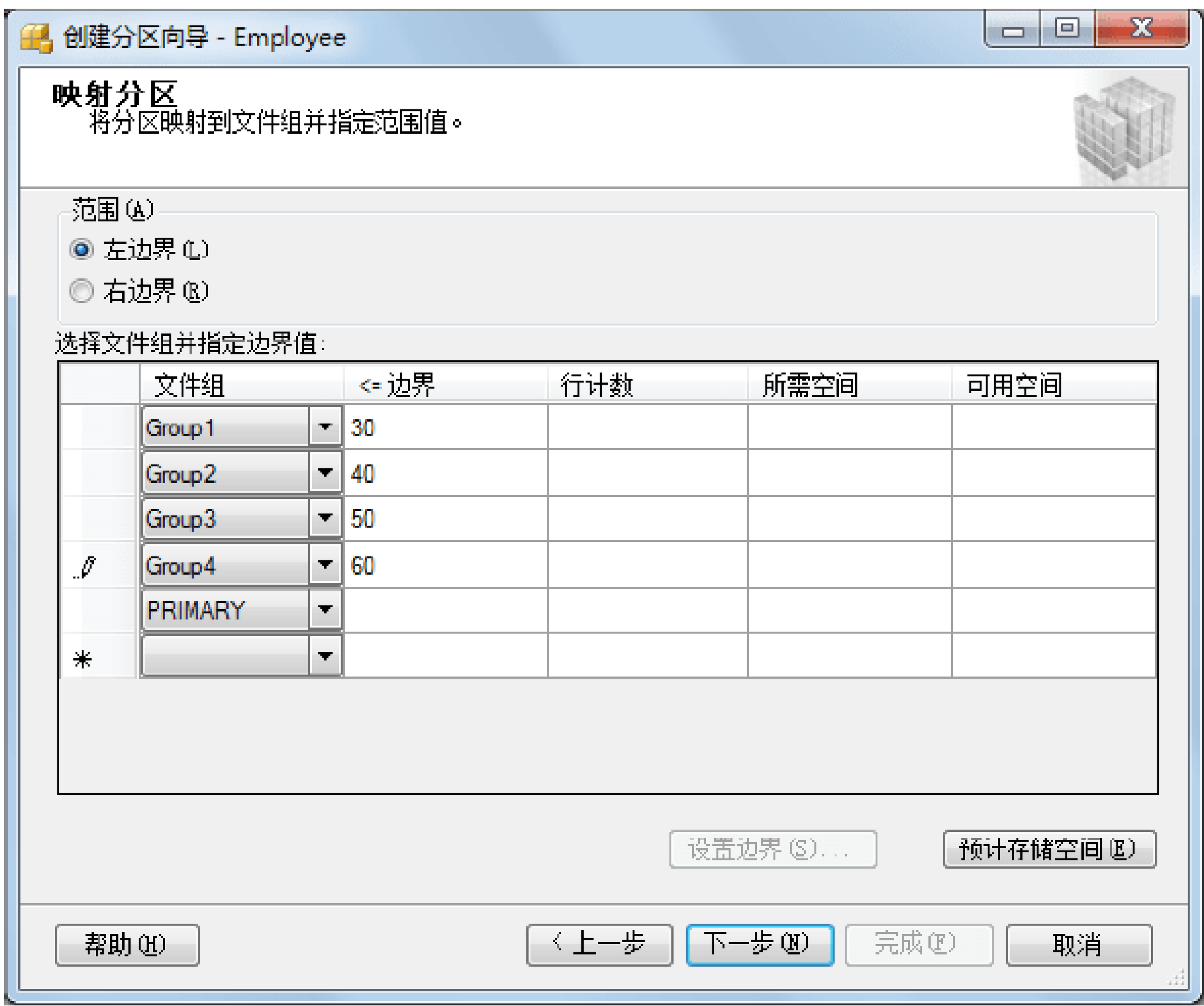


图 5.9　选择文件组合指定边界值

（9）弹出“选择输出选项”界面，选中“立即运行”单选按钮，然后单击“完成”按钮，完成对 Employee 表的分区操作，如图 5.10 所示。

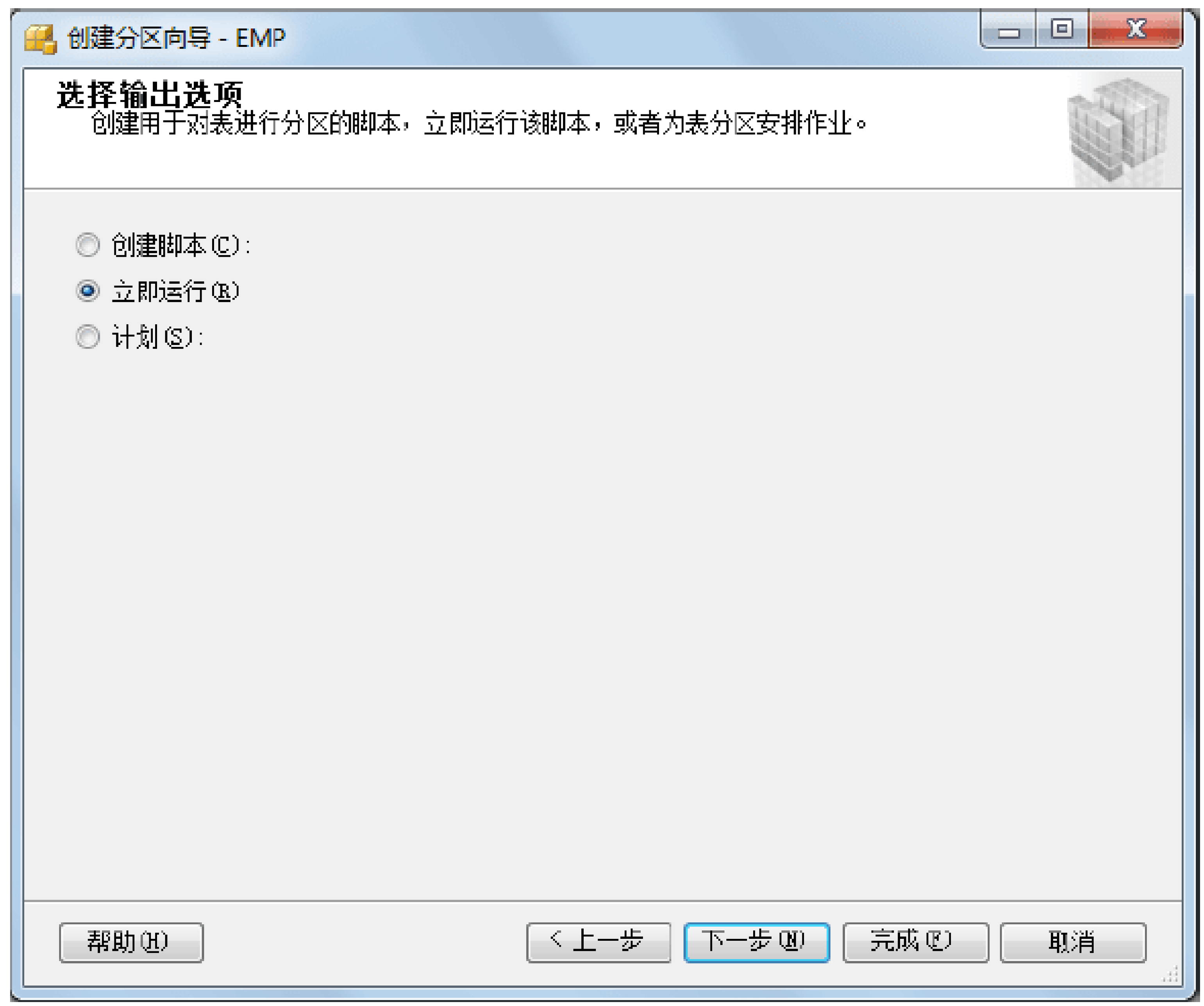


图 5.10　选择输出选项

（10）单击“完成”按钮之后，会出现如图 5.11 所示的界面，再次单击“完成”按钮。



虽然分区可以带来很多好处，但是也会增加实现对象的管理操作和复杂性。所以，可能不需要为较小的表或目前满足性能和维护要求的表分区。本书中所涉及的表都是较小的表，所以不必建立分区表。

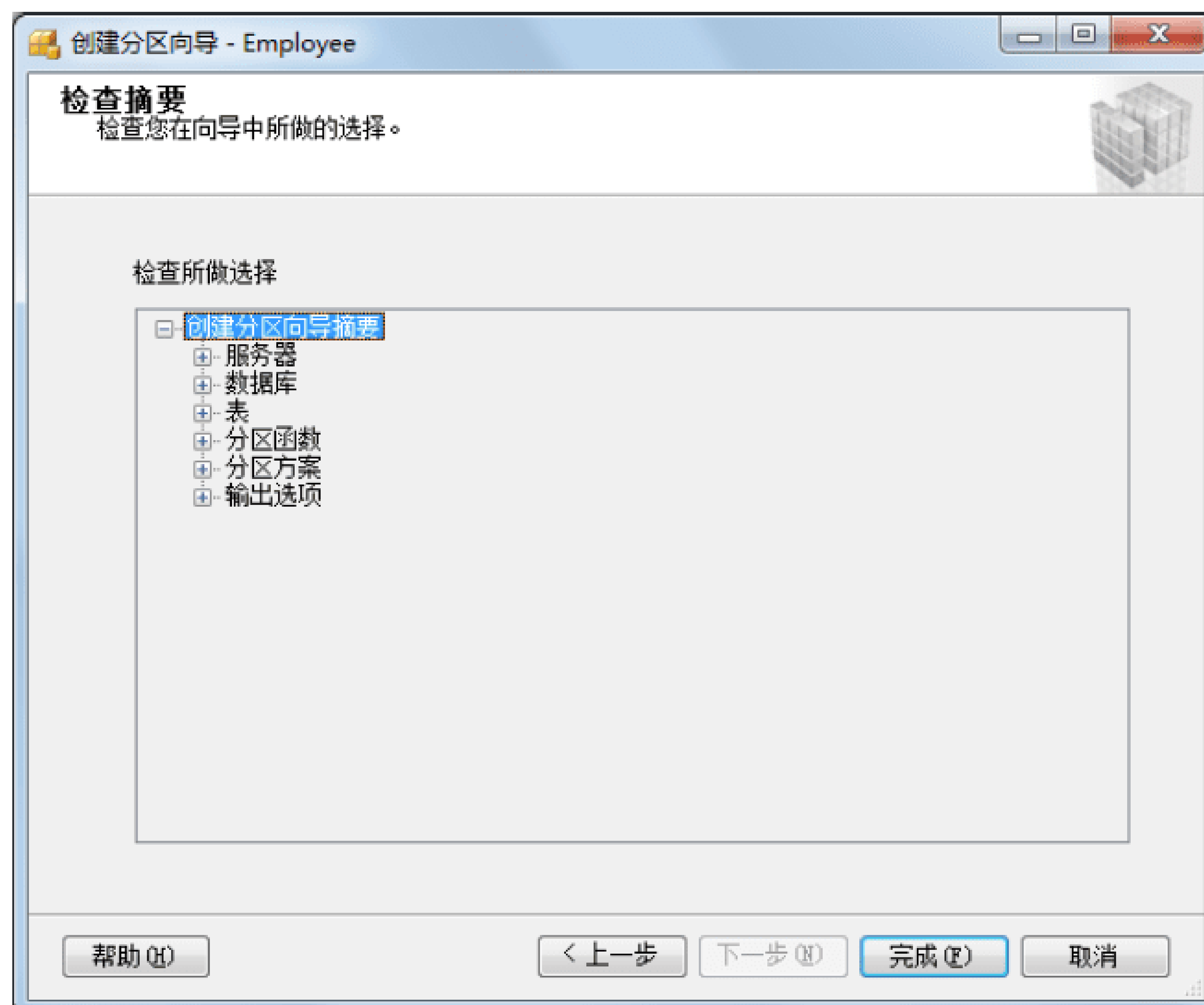


图 5.11　创建分区

5.1.3　命令创建分区表

1. 创建分区函数

创建分区函数的语法格式如下：

```
CREATE PARTITION FUNCTION partition_function_name (input_parameter_type)
AS RANGE [LEFT | RIGHT]
FOR VALUES ([boundary_value [,…n]])
[;]
```

参数说明如下。

☑ partition_function_name：是要创建的分区函数的名称。

☑ input_parameter_type：用于分区的列的数据类型。

☑ LEFT | RIGHT：指定当间隔值由数据库引擎按升序从左到右排列时，boundary_value 属于每个边界值间隔的那一侧（左侧或者右侧）。如果未指定，则默认值为 LEFT。

☑ boundary_value：为使用 partition_function_name 的已分区表或索引的每个分区指定边界值。如果为空，则分区函数使用 partition_function_name 将整个表或索引映射到单个分区。boundary_value 是可以引用变量的常量表达式。boundary_value 必须与 input_parameter_type 中提供的数据类型相匹配，或者可隐式转换为该数据类型。[,…n]指定 boundary_value 提供的值的数目，不能超过 999。所创建的分区数等于 n+1。

【例 5.01】　对 int 类型的列创建一个名为 AgePF 的分区函数，该函数把 int 类型的列中数据分成 6 个区。分为小于或等于 10 的区、大于 10 且小于或等于 30 的区、大于 30 且小于或等于 50 的区、大于 50 且小于或等于 70 的区、大于 70 且小于或等于 80 的区、大于 80 的区。（**实例位置：资源包\源码**

05\5.01）

代码如下：

```
CREATE PARTITION FUNCTION AgePF (int)
 AS RANGE LEFT FOR VALUES (10,30,50,80)
GO
```

2. 创建分区方案

分区函数创建完后，使用 CREATE PARTITION SCHEME 命令创建分区方案，由于在创建分区方案时需要根据分区函数的参数定义映射分区的文件组。所以需要有文件组来容纳分区数，文件组可以由一个或多个文件构成，而每个分区必须映射到一个文件组中。一个文件组可以由多个分区使用。通常情况下，文件组的数目最好与分区数目相同，并且这些文件组通常位于不同的磁盘上。一个分区方案只可以使用一个分区函数，而一个分区函数可以用于多个分区方案中。

创建分区方案的语法格式如下：

```
CREATE PARTITION SCHEME partition_scheme_name
AS PARTITION partition_function_name
[ALL] TO ({file_group_name | [PRIMARY]} [,…n])
[;]
```

参数说明如下。

☑ partition_scheme_name：创建的分区方案的名称，在创建表时使用该方案可以创建分区表。

☑ partition_function_name：使用分区方案的分区函数的名称，该函数必须在数据库中存在，分区函数所创建的分区将映射到在分区方案中指定的文件组。单个分区不能同时包含 FILESTREAM 和非 FILESTREAM 文件组。

☑ ALL：指定所有分区都映射到在 file_group_name 中提供的文件组，或映射到主文件组（如果指定了[PRIMARY]）。如果指定了 ALL，则只能指定一个 file_group_name。

☑ file_group_name：指定用来持有由 partition_function_name 指定的分区的文件组的名称。分区分配到文件组的顺序是从分区 1 开始，按文件组在[,…n]中列出的顺序进行分配。在[,…n]中，可以多次指定同一个 file_group_name。

【例 5.02】 假如数据库 db_2012 中存在 FGroup1、FGroup2、FGroup3、FGroup4、FGoup5、FGroup6 这 6 个文件组，根据例 5.01 中定义的分区函数创建一个分区方案，将分区函数中的 6 个分区分别存放在这 6 个文件组中。（**实例位置：资源包\源码\05\5.02**）

代码如下：

```
CREATE PARTITION SCHEME AgePS
AS PARTITION AgePF
TO (FGroup1,FGroup2,FGroup3,FGroup4,FGoup5,FGroup6)
GO
```

3. 使用分区方案创建分区表

分区函数和分区方案创建完后就可以创建分区表了。创建分区表使用 CREATE TABLE 语句，只要

在 ON 关键字的后面指定分区方案和分区列即可。

【例 5.03】 在数据库 db_2012 中创建分区表，表中包含 ID、“姓名”和“年龄”（年龄取值范围是 1～100），使用例 5.02 的方案。（**实例位置：资源包\源码\05\5.03**）

代码如下：

```
CREATE TABLE sample
(
ID int NOT NULL,
姓名 varchar(8) NOT NULL,
年龄 int NOT NULL
)
ON AgePS(年龄)
GO
```



已分区表的分区列在数据类型、长度、精度与分区方案索引用的分区函数使用的数据类型、长度、精度要一致。



5.2　操作表数据

5.2.1　使用 SQL Server Management Studio 添加记录

打开数据表后，在最后一条记录下面有一条所有字段都为 NULL 的记录，在此条记录中添加新记录。向数据表（如 student）中添加数据的具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库。

（3）选择数据表 student，单击鼠标右键，在弹出的快捷菜单中选择“编辑前 200 行”命令，如图 5.12 所示。

（4）进入数据表编辑窗口，最后一条记录下面有一条所有字段都为 NULL 的记录，如图 5.13 所示，在此处添加新记录。记录添加后数据将自动保存在数据表中。

在新增记录内容时有以下几点需要注意。

（1）设置为标识规范的字段不能输入字段内容。

（2）被设置为主键的字段不允许与其他行的主键值相同。

（3）输入字段内容的数据类型和字段定义的数据类型一致，包括数据类型、长度和精度等。

（4）不允许 NULL 的字段必须输入与字段类型的数据。

（5）作为外键的字段，输入的内容一定要符合外键要求。

（6）如果字段存在其他约束，输入的内容必须满足约束要求。

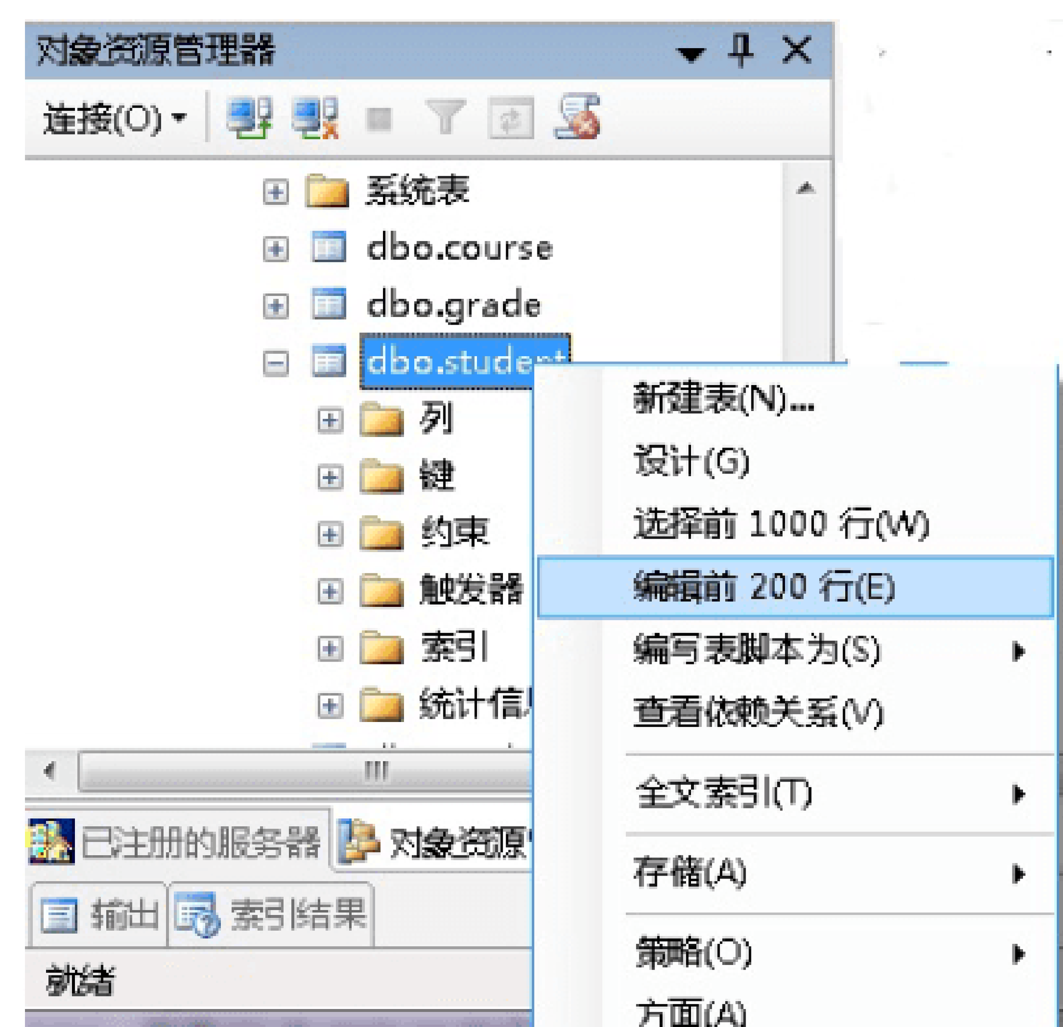


图 5.12　打开数据表

	学号	姓名	性别
▶	B001	李艳丽	女
	B002	聂乐乐	女
	B003	刘大伟	男
	B004	王嘟嘟	女
	B005	李羽凡	男
	B006	刘月	女
*	NULL	NULL	NULL

图 5.13　编辑窗口

（7）如果字段被设置默认值，当不在字段内输入任何数据时会字段填入默认值。

5.2.2　使用 INSERT 语句添加记录

使用 INSERT 语句可以向数据表插入记录，INSERT 语句可以在查询编辑器窗口中执行。本小节将对 INSERT 语句的执行进行讲解。

1．INSERT 语句的语法

Transact-SQL 中 INSERT 语句的基本语法格式如下：

```
INSERT INTO 表名[(列名 1，列名 2，列名 3...)] VALUES(值 1，值 2，值 3...)
```

或：

```
INSERT INTO 表名[(列名 1，列名 2，列名 3...)] SELECT 语句
```

2．INSERT 语句添加数据的实例

使用 INSERT 语句向员工基本信息表中插入记录，代码如下：

```
INSERT INTO tb_basicMessage VALUES('小李',26,'男',4,4)
```

语句执行后数据表记录如图 5.14 所示。

	id	name	age	sex	dept	headship
1	7	小陈	27	男	1	1
2	8	小葛	29	男	1	1
3	16	张三	30	男	1	5
4	23	小开	30	男	4	4
5	24	金额	20	女	4	7
6	25	cdd	24	女	3	6
7	27	———	25	男	2	3
8	29	小李	26	男	4	4

图 5.14　插入后的数据

5.2.3　使用 SQL Server Management Studio 修改记录

使用 SQL Server Management Studio 打开数据表后，可以在需要修改的字段的单元格内修改字段内容。数据表中错误或过时的数据记录可以进行修改。修改数据表中数据记录的具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 数据库中。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库。

（3）选择数据表 student，单击鼠标右键，在弹出的快捷菜单中选择“编辑前 200 行”命令。

（4）进入数据表编辑窗口，如图 5.15 所示，直接单击需要修改字段的单元格，对数据进行修改。

	学号	姓名	性别
	B005	李羽凡	男
	B006	刘月	女
✎	B007	高兴	男
*	NULL	NULL	NULL

图 5.15　使用图形界面修改数据

5.2.4　使用 UPDATE 语句修改记录

使用 UPDATE 语句可以向数据表插入记录，UPDATE 语句可以在查询编辑器窗口中执行。本小节将对 UPDATE 语句的执行进行讲解。

1. UPDATE 语句的语法

Transact-SQL 中 UPDATE 语句的基本语法格式如下：

```
UPDATE 表名 SET 列名 1 = 值 1 [，列名 2=值 2，列名 3=值 3...] [WHERE 子句]
```

2. UPDATE 语句更新数据的实例

【例 5.04】　使用 UPDATE 语句更新所有记录。（**实例位置：资源包\源码\05\5.04**）

使用 UPDATE 语句将数据表 tb_basicMessage 中所有数据的 sex 字段值都改为“男”，代码如下：

```
UPDATE tb_basicMessage SET sex='男'
```

修改的数据如图 5.16 所示。

	id	name	age	sex	dept	headship
1	7	小陈	27	男	1	1
2	8	小葛	29	男	1	1
3	16	张三	30	男	1	5
4	23	小开	30	男	4	4
5	24	金额	20	男	4	7
6	25	cdd	24	男	3	6
7	27	———	25	男	2	3
8	29	小李	26	男	4	4

图 5.16　更新记录后

【例 5.05】 使用 UPDATE 语句更新符合条件的记录。（**实例位置：资源包\源码\05\5.05**）

将姓名为“刘大伟”的人员性别设置为“男”，代码如下：

```
UPDATE student SET 性别='男' WHERE 姓名='刘大伟'
```

语句执行后，数据表的记录如图 5.17 所示。

	学号	姓名	性别	年龄	出生日期	联系方式
1	B001	李艳丽	女	25	1985-03-03	13451
2	B002	聂乐乐	女	23	1984-03-10	23451
3	B003	刘大伟	男	23	1986-01-01	52345
4	B004	王嘟嘟	女	22	1984-03-10	62345

图 5.17 修改指定记录后

5.2.5 使用 SQL Server Management Studio 删除记录

使用 SQL Server Management Studio 打开数据表后，选中要删除的记录，单击鼠标右键，在弹出的快捷菜单中选择“删除”命令，如图 5.18 所示。

将数据表 Table_1 中的记录进行删除，具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定数据库。

（3）选择数据表 student，单击鼠标右键，在弹出的快捷菜单中选择“编辑前 200 行”命令。

（4）进入数据表编辑窗口，选中要删除的数据记录，单击鼠标右键，在弹出的快捷菜单中选择“删除”命令。

（5）在弹出的提示对话框中，单击“是”按钮即可删除该记录，如图 5.19 所示。

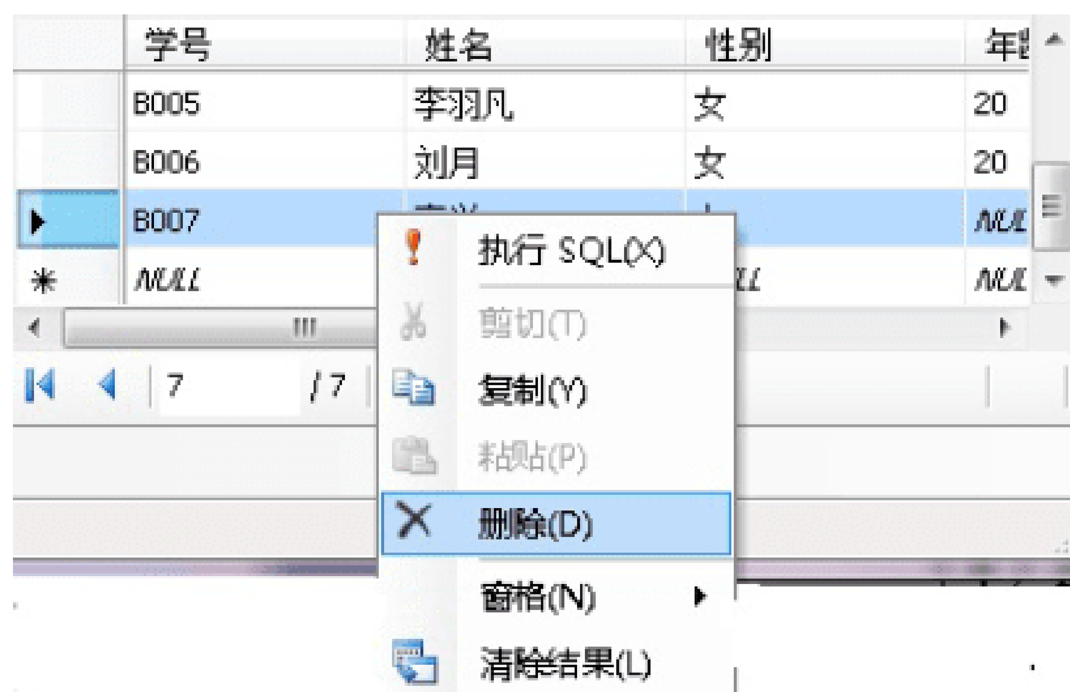


图 5.18 图形界面删除数据



图 5.19 提示删除对话框

5.2.6 使用 DELETE 语句删除记录

使用 DELETE 语句也可以删除表中的记录，本小节将对 DELETE 语句的执行进行讲解。

1. DELETE 语句的语法

Transact-SQL 中 DELETE 语句的基本语法格式如下：

```
DELETE [FROM] 表名 [WHERE 子句]
```

2．DELETE 语句删除数据的实例

【例 5.06】 使用 DELETE 语句删除指定记录。（**实例位置：资源包\源码\05\5.06**）

删除表 tb_basicMessage 中的 name 为“小李”的记录，代码如下：

```
DELETE tb_basicMessage WHERE name='小李'
```

删除前数据表中的记录如图 5.20 所示，删除后数据表中的记录如图 5.21 所示。

	id	name	age	sex	dept	headship
1	7	小陈	27	男	1	1
2	8	小葛	29	男	1	1
3	16	张三	30	男	1	5
4	23	小开	30	男	4	4
5	24	金颤	20	男	4	7
6	25	cdd	24	男	3	6
7	27	———	25	男	2	3
8	29	小李	26	男	4	4

图 5.20　删除数据前

	id	name	age	sex	dept	headship
1	7	小陈	27	男	1	1
2	8	小葛	29	男	1	1
3	16	张三	30	男	1	5
4	23	小开	30	男	4	4
5	24	金颤	20	男	4	7
6	25	cdd	24	男	3	6
7	27	———	25	男	2	3

图 5.21　删除数据后

如果 DELETE 语句中不包含 WHERE 子句，则将删除全部记录。例如：

```
DELETE student
```

5.3　表与表之间的关联



视频讲解

关系是通过匹配键列中的数据而工作的，而键列通常是两个表中具有相同名称的列，在数据表间创建关系可以显示某个表中的列连接到另一个表中的列。表与表之间存在 3 种类型的关系，所创建的关系类型取决于相关联的列是如何定义的。表与表之间存在如下 3 种关系。

☑　一对一关系。

☑　一对多关系。

☑　多对多关系。

5.3.1　一对一关系

一对一关系是指表 A 中的一条记录确实在表 B 中有且只有一条相匹配的记录。在一对一关系中，大部分相关信息都在一个表中。

如果两个相关列都是主键或具有唯一约束，创建的就是一对一关系。

在学生管理系统中，Course 表用于存放课程的基础信息，这里定义为主表；teacher 表用于存放教师信息，这里定义为从表，且一个教师只能教一门课程。下面介绍如何通过这两张表创建一对一关系。



> **说明**
>
> “一个教师只能教一门课程”，在这里不考虑一名教师教多门课程的情况。例如，英语专业的老师只能教英语。

操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）鼠标右键单击 Course 表，在弹出的快捷菜单中选择“设计”命令。

（4）在表设计窗口中，鼠标右键单击 Cno 字段，在弹出的快捷菜单中选择“关系”命令，打开“外键关系”窗体，在该窗体中单击“添加”按钮，如图 5.22 所示。

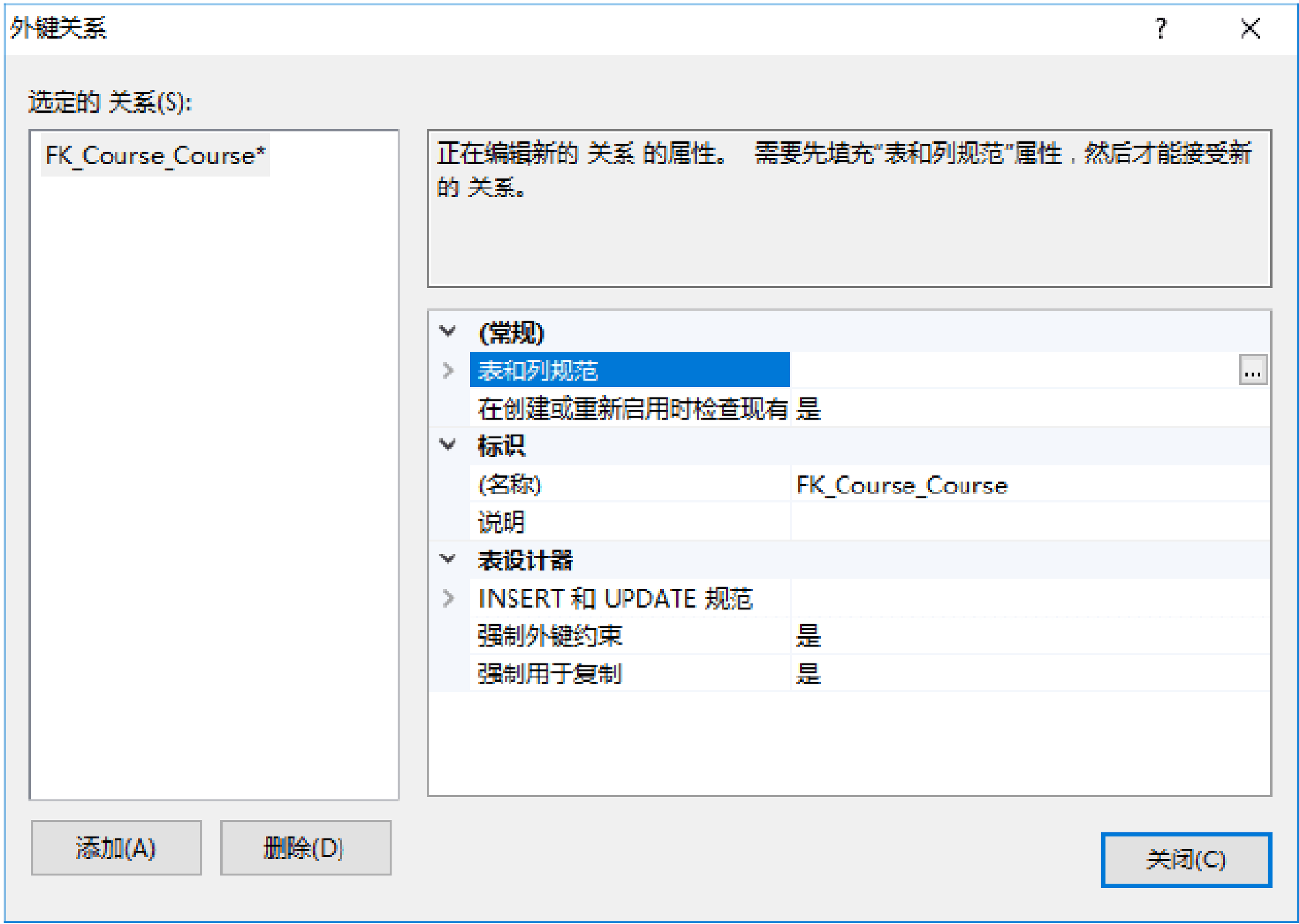


图 5.22 “外键关系”窗体

（5）在“外键关系”窗体中，单击“表和列规范”后面的…按钮，添加表和列规范属性，弹出“表和列”窗体，在该窗体中设置关系名及主外键的表，如图 5.23 所示。

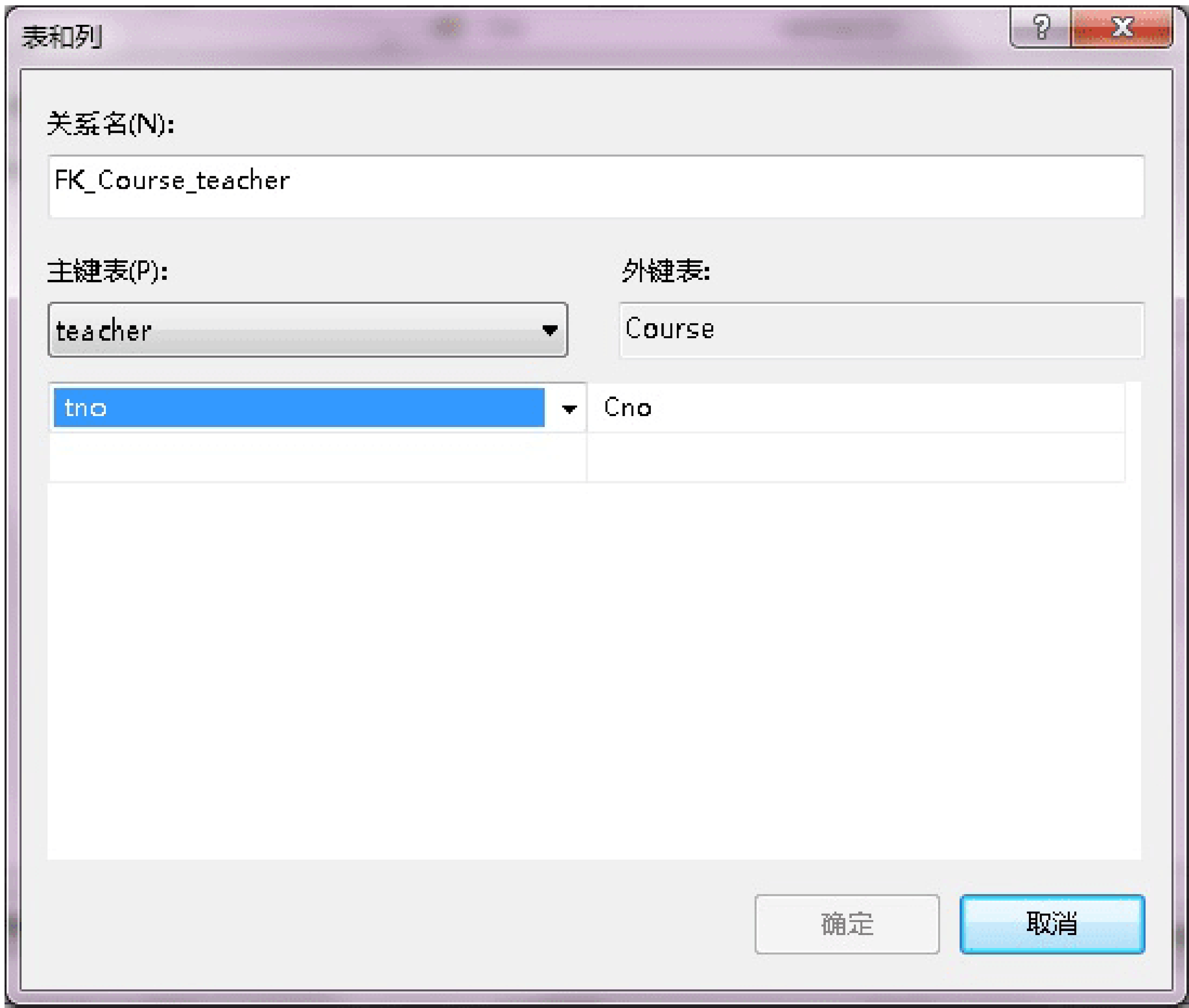


图 5.23 “表和列”窗体

（6）在“表和列”窗体中，单击“确定”按钮，返回到“外键关系”窗体，在“外键关系”窗体

中单击“关闭”按钮，完成一对一关系的创建。



创建一对一关系之前，都应将 tno、Cno 设置为这两个表的主键，且关联字段类型必须相同。

5.3.2　一对多关系

一对多关系是最常见的关系类型，是指表 A 中的行可以在表 B 中有许多匹配行，但是表 B 中的行只能在表 A 中有一个匹配行。

如果在相关列中只有一列是主键或具有唯一约束，则创建的是一对多关系。例如，student 用于存储学生的基础信息，这里定义为主表；Course 用于存储课程的基础信息，一个学生可以学多门课程，这里定义为从表。下面介绍如何通过这两张表创建一对多关系。

操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）鼠标右键单击 Course 表，在弹出的快捷菜单中选择“设计”命令。

（4）在表设计窗口中，鼠标右键单击 Cno 字段，在弹出的快捷菜单中选择“关系”命令，打开“外键关系”窗体，在该窗体中单击“添加”按钮，如图 5.24 所示。

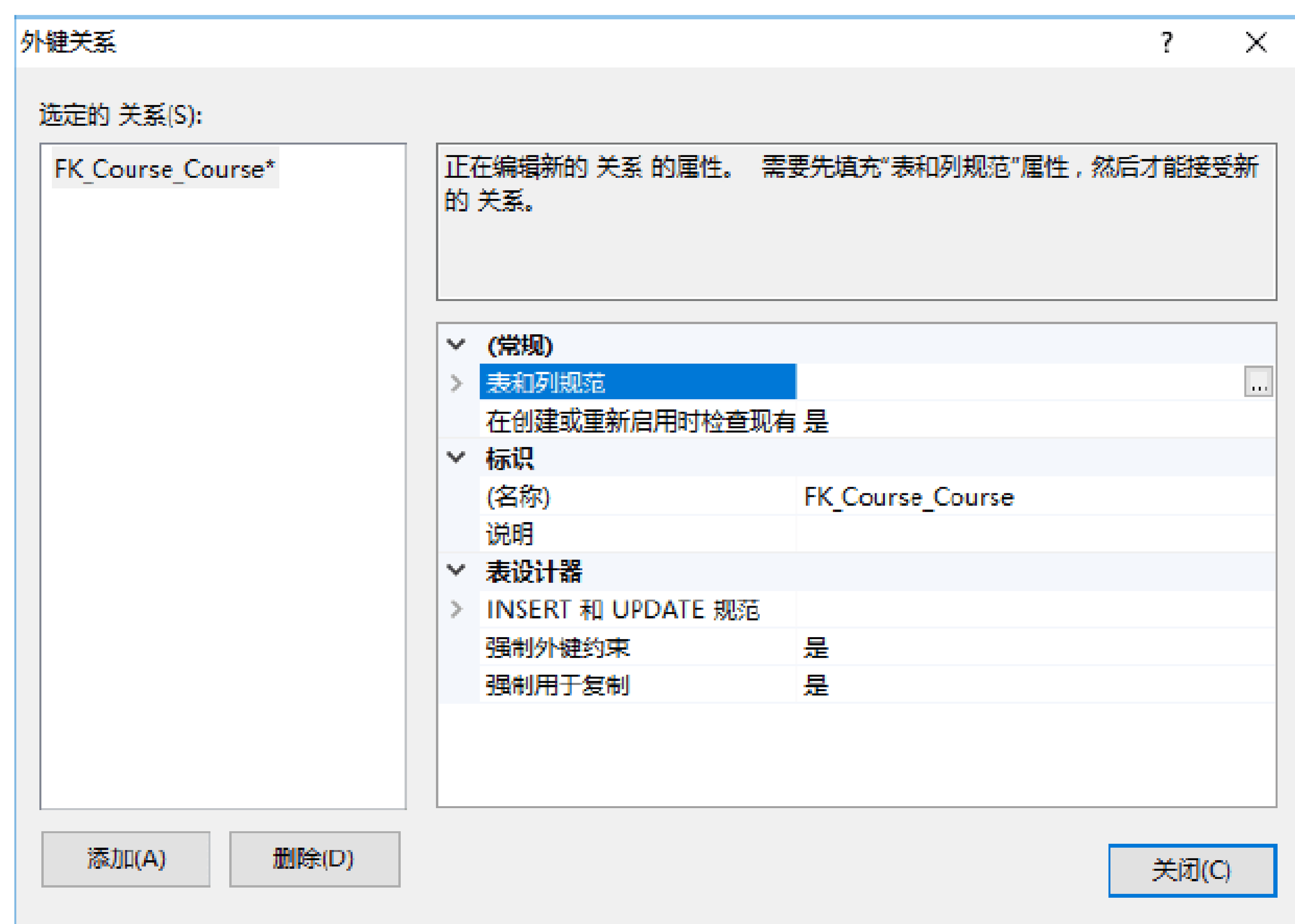


图 5.24　“外键关系”窗体

（5）在“外键关系”窗体中，单击“表和列规范”后面的…按钮，选择要创建一对多关系的数据表和列。弹出“表和列”窗体，在该窗体中设置关系名及主外键的表，如图 5.25 所示。

（6）在“表和列”窗体中，单击“确定”按钮，返回到“外键关系”窗体，在“外键关系”窗体中单击“关闭”按钮，完成一对多关系的创建。

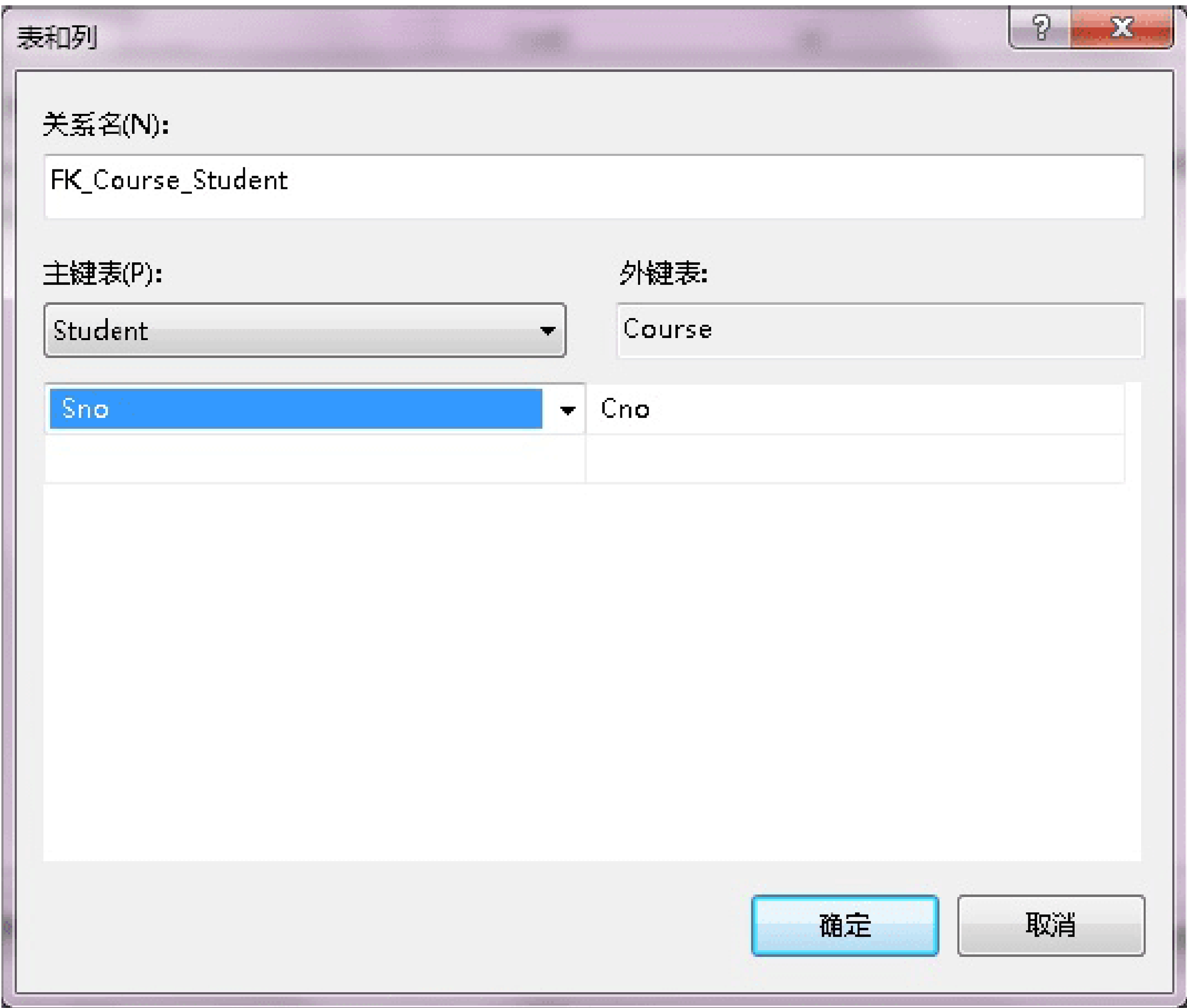


图 5.25 “表和列”窗体

5.3.3 多对多关系

多对多关系是指关系中每个表的行在相关表中具有多个匹配行。在数据库中，多对多关系的建立是依靠第 3 个表即连接表实现的，连接表包含相关的两个表的主键列，然后从两个相关表的主键列分别创建与连接表中匹配列的关系。

例如，通过“商品信息表”与“商品订单表”创建多对多关系。首先就需要建立一个连接表（如“商品订单信息表”），该表中应该包含上述两个表的主键列，然后“商品信息表”和“商品订单表”分别与连接表建立一对多关系，以此来实现“商品信息表”和“商品订单表”的多对多关系。

5.4 小　　结

本章介绍操作表数据的方法，分为使用 SQL Server Management Studio 和命令方式。通过本章的学习了解了分区表的创建、数据表中数据的添加、修改和删除，最后了解了表与表之间的关联。

第 6 章

SQL 函数的使用

（视频讲解：42 分钟）

在 SQL Server 中提供了许多内置函数，按函数种类可以分为聚合函数、数学函数、字符串函数、日期和时间函数、转换函数、元数据函数等 6 种。在进行查询操作时，经常能够用到 SQL 函数，使用 SQL 函数会给查询带来很多方便。本章将会对不同类型的 SQL 函数进行讲解，从而使读者能够快速地掌握好 SQL 函数的使用方法。

学习摘要：

- SQL 函数的几种主要分类
- 常用聚合函数
- 常用数学函数
- 常用字符串函数
- 常用日期和时间函数
- 转换函数
- 常用元数据函数



6.1 聚 合 函 数

聚合函数对一组值执行计算，并返回单个值。除了 COUNT 以外，聚合函数都会忽略空值。聚合函数经常与 SELECT 语句的 GROUP BY 子句一起使用。

所有聚合函数均为确定性函数。这表示任何时候使用一组特定的输入值调用聚合函数，所返回的值都是相同的。

6.1.1 聚合函数概述

聚合函数对一组值进行计算并返回单一的值，通常聚合函数会与 SELECT 语句的 GROUP BY 子句一同使用，在与 GROUP BY 子句使用时，聚合函数会为每一个组产生一个单一值，而不会为整个表产生一个单一值。常用的聚合函数及说明如表 6.1 所示。

表 6.1 常用的聚合函数及说明

函数名称	说明
SUM	返回表达式中所有值的和
AVG	计算平均值
MIN	返回表达式的最小值
MAX	返回表达式的最大值
COUNT	返回组中项目的数量
DISTINCT	返回一个集合，并从指定集合中删除重复的元组

6.1.2 SUM（求和）函数

SUM 函数返回表达式中所有值的和或仅非重复值的和。SUM 只能用于数字列。空值将被忽略。

语法格式如下：

```
SUM([ALL | DISTINCT] expression)
```

参数说明如下。

☑ ALL：对所有的值应用此聚合函数。ALL 是默认值。

☑ DISTINCT：指定 SUM 返回唯一值的和。

☑ expression：常量、列或函数与算术、位和字符串运算符的任意组合。expression 是精确数字或近似数字数据类型类别（bit 数据类型除外）的表达式。

☑ 返回类型：以最精确的 expression 数据类型返回所有 expression 值的和。

有关 SUM 函数使用的几点说明如下。

☑ 含有索引的字段能够加快聚合函数的运行。

- ☑ 字段数据类型为 int、smallint、tinyint、decimal、numeric、float、real、money 以及 smallmoney 的字段才可以使用 SUM 函数。
- ☑ 在使用 SUM 函数时，SQL Server 把结果集中的 smallint 或 tinyint 这些数据类型当作 int 处理。
- ☑ 在使用 SUM 函数时，SQL Server 将忽略空值（NULL），即计算时不计算这些空值。

【例 6.01】 使用 SUM 函数，求 SC 表中 001（数据结构）课程的总成绩，SQL 语句及运行结果如图 6.1 所示。（**实例位置：资源包\源码\06\6.01**）

SQL 语句如下：

```
USE db_2014
SELECT SUM(Grade) AS 数据结构总成绩
FROM SC WHERE Cno=001
```

在 SC 表中 001 的成绩如图 6.2 所示。

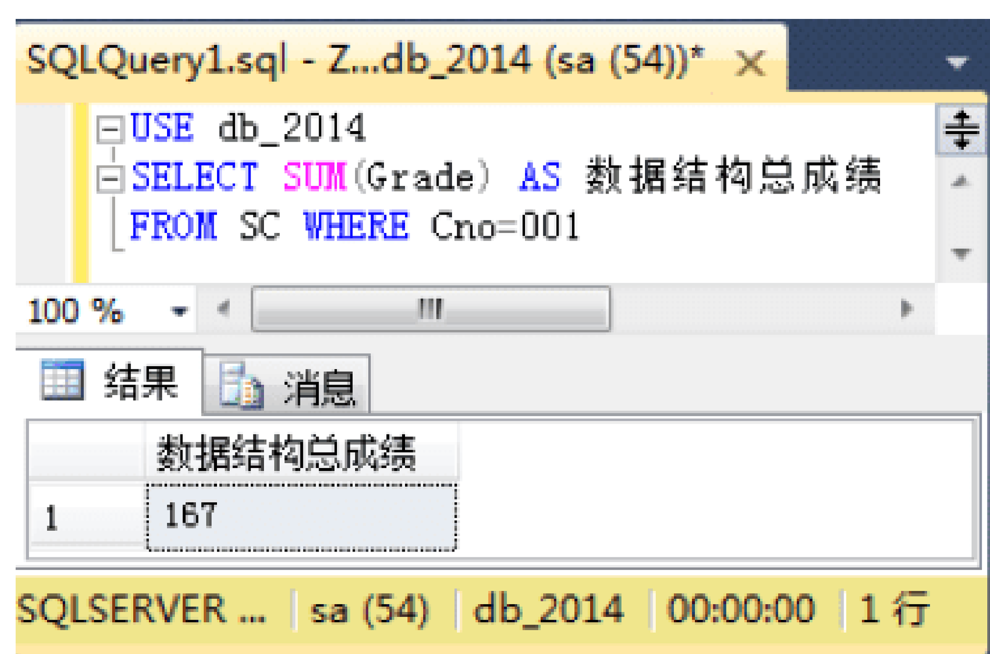


图 6.1　使用 SUM 函数获得数据结构的总成绩

结果　消息

	Sno	Cno	Grade
1	201109001	001	93
2	201109002	002	70
3	201109003	001	74
4	201109004	001	NULL
5	201109005	002	90

图 6.2　SC 表中 001 的成绩

6.1.3　AVG（平均值）函数

AVG 函数返回组中各值的平均值。将忽略空值。

语法格式如下：

```
AVG([ALL | DISTINCT] expression)
```

参数说明如下。

- ☑ ALL：对所有的值进行聚合函数运算。ALL 是默认值。
- ☑ DISTINCT：指定 AVG 只在每个值的唯一实例上执行，而不管该值出现了多少次。
- ☑ expression：是精确数值或近似数值数据类别（bit 数据类型除外）的表达式。不允许使用聚合函数和子查询。
- ☑ 返回类型：返回类型由 expression 的计算结果类型确定。

有关 AVG 函数使用的几点说明如下。

- ☑ AVG 函数不一定返回与传递到函数的列完全相同的数据类型。
- ☑ AVG 函数只能用于数据类型是 int、smallint、tinyint、decimal、float、real、money 和 smallmoney 的字段。
- ☑ 在使用 AVG 函数时，SQL Server 把结果集中的 smallint 或 tinyint 这些数据类型当作 int 处理。

AVG 函数的返回值类型由表达式的运算结果类型决定，如表 6.2 所示。

表 6.2 AVG 函数返回值类型

表达式结果	返 回 类 型
整数分类	int
decimal 分类(p,s)	decimal(38,s)除以 decimal(10,0)
money 和 smallmoney 分类	money
float 和 read 分类	float

【例 6.02】 使用 AVG 函数，求 SC 表中 001（数据结构）课程的平均成绩，SQL 语句及运行结果如图 6.3 所示。（**实例位置：资源包\源码\06\6.02**）

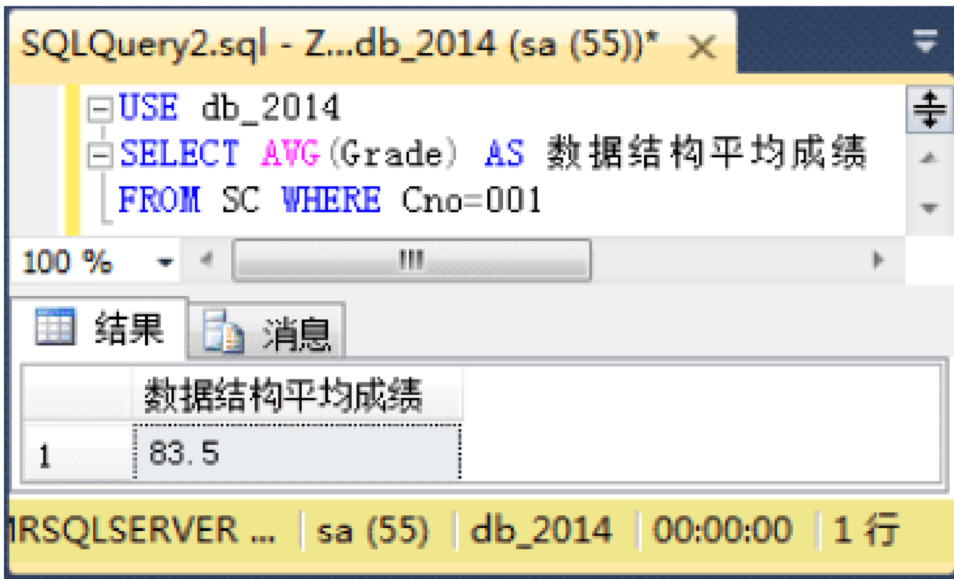


图 6.3 使用 AVG 函数获得数据结构的平均成绩

SQL 语句如下：

```
USE db_2014
SELECT AVG(Grade) AS 数据结构平均成绩
FROM SC WHERE Cno=001
```

6.1.4 MIN（最小值）函数

MIN 函数返回表达式中的最小值。

语法格式如下：

```
MIN([ALL | DISTINCT] expression)
```

参数说明如下。

- ☑ ALL：对所有的值进行聚合函数运算。ALL 是默认值。
- ☑ DISTINCT：指定每个唯一值都被考虑。DISTINCT 对于 MIN 无意义，使用它仅仅是为了符合 ISO 标准。
- ☑ expression：常量、列名、函数以及算术运算符、位运算符和字符串运算符的任意组合。MIN 可用于 numeric、char、varchar 或 datetime 列，但不能用于 bit 列。不允许使用聚合函数和子查询。
- ☑ 返回类型：返回与 expression 相同的值。

有关 MIN 函数使用的几点说明如下。

- ☑ MIN 函数不能用于数据类型是 bit 的字段。
- ☑ 在确定列中的最小值时，MIN 函数忽略 NULL 值，但是如果在该列中的所有行都有 NULL 值，

将返回 NULL 值。

☑ 不允许使用聚合函数和子查询。

【例 6.03】 使用 MIN 函数，查询 Student 表中男同学的最小年龄，SQL 语句及运行结果如图 6.4 所示。（**实例位置：资源包\源码\06\6.03**）

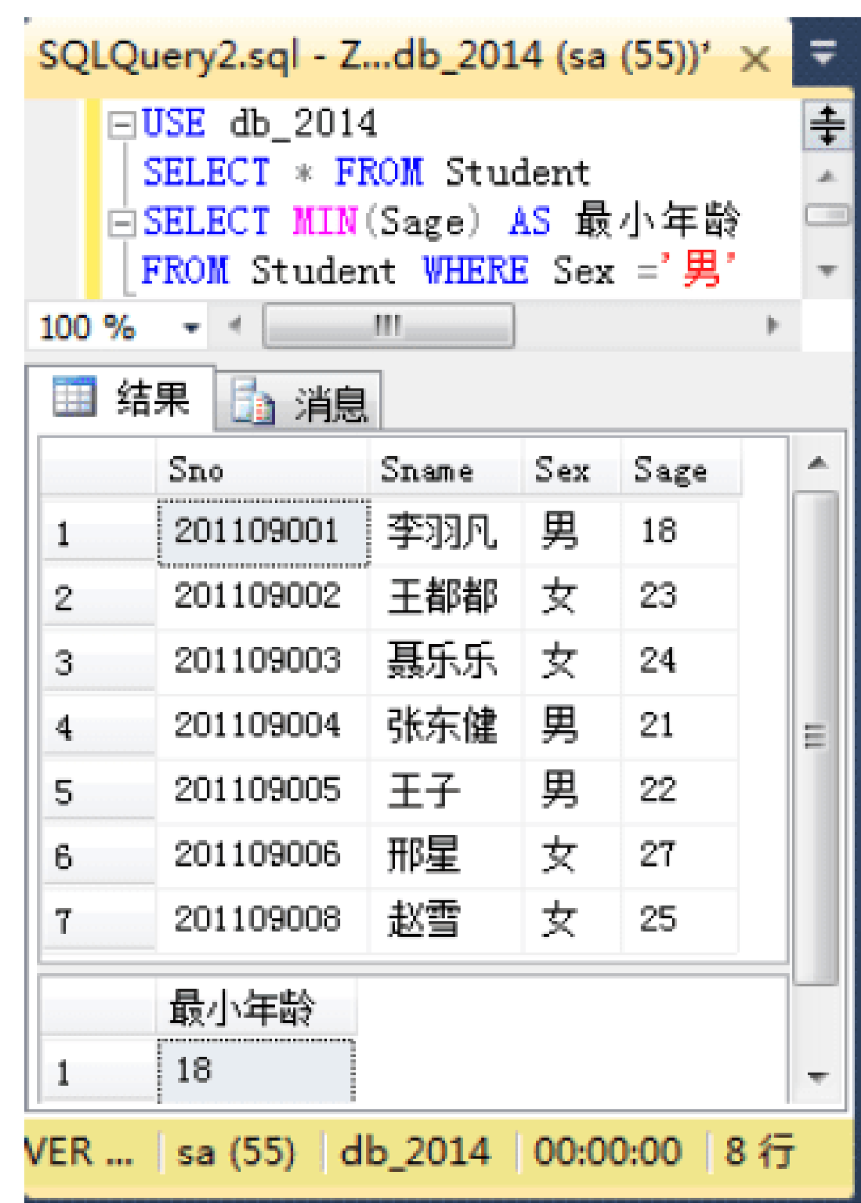


图 6.4　使用 MIN 函数获得数据结构的最小成绩

SQL 语句如下：

```
USE db_2014
SELECT * FROM Student
SELECT MIN(Sage) AS 最小年龄
FROM Student WHERE Sex ='男'
```

6.1.5　MAX（最大值）函数

MAX 函数返回表达式的最大值。

语法格式如下：

```
MAX([ALL | DISTINCT] expression)
```

参数说明如下。

☑ ALL：对所有的值应用此聚合函数。ALL 是默认值。

☑ DISTINCT：指定考虑每个唯一值。DISTINCT 对于 MAX 无意义，使用它仅仅是为了与 ISO 实现兼容。

☑ expression：常量、列名、函数以及算术运算符、位运算符和字符串运算符的任意组合。MAX 可用于 numeric 列、character 列和 datetime 列，但不能用于 bit 列。不允许使用聚合函数和子查询。

☑ 返回类型：返回与 expression 相同的值。

有关 MAX 函数使用的几点说明如下。

☑ MAX 函数将忽略选取对象中的空值。

☑ 不能通过 MAX 函数从 bit、text 和 image 数据类型的字段中选取最大值。

☑ 在 SQL Server 中，MAX 函数可以用于数据类型为数字、字符、datetime 的列，但是不能用于数据类型为 bit 的列。不能使用聚合函数和子查询。

☑ 对于字符列，MAX 查找排序序列的最大值。

【例 6.04】 在本示例中使用了一个子查询，并在子查询中使用了 MAX 函数将查询条件指定为 Student 表中年龄最大的同学信息，SQL 语句及运行结果如图 6.5 所示。（**实例位置：资源包\源码\06\6.04**）

SQLQuery2.sql - Z...db_2014 (sa (55))*

```
SELECT * FROM Student
SELECT Sname,Sex,Sage FROM Student
WHERE Sage=(SELECT MAX(Sage) FROM Student)
```

100 %

结果　消息

	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	18
2	201109002	王都都	女	23
3	201109003	聂乐乐	女	24
4	201109004	张东健	男	21
5	201109005	王子	男	22
6	201109006	邢星	女	27
7	201109008	赵雪	女	25

	Sname	Sex	Sage
1	邢星	女	27

G-PC\MRSQLSERVER ... | sa (55) | db_2014 | 00:00:00 | 8 行

图 6.5　使用 MAX 函数获取 Student 表中年龄最大的同学信息

SQL 语句如下：

```
USE db_2014
SELECT * FROM Student
SELECT Sname,Sex,Sage FROM Student
WHERE Sage=(SELECT MAX(Sage) FROM Student)
```

首先在 Student 表中选择指定列的数据并显示，然后在 WHERE 条件中使用子查询，并在子查询中使用 MAX 函数选择 Student 中年龄最大的同学。

如果用户不想获取其他列的信息，可以直接在 SELECT 语句中使用 MAX 函数加上要查询的列即可。

【例 6.05】 直接查询学生中年龄最大的同学。（**实例位置：资源包\源码\06\6.05**）

SQL 语句如下：

```
USE db_2014
SELECT MAX(Sage) AS 最大年龄 FROM Student
```

6.1.6　COUNT（统计）函数

COUNT 函数返回组中的项数。COUNT 返回 int 数据类型值。

语法格式如下：

```
COUNT({[[ALL | DISTINCT] expression] | *})
```

参数说明如下。

☑ ALL：对所有的值进行聚合函数运算。ALL 是默认值。

☑ DISTINCT：指定 COUNT 返回唯一非空值的数量。

☑ expression：除 text、image 或 ntext 以外任何类型的表达式。不允许使用聚合函数和子查询。

☑ *：指定应该计算所有行以返回表中行的总数。COUNT(*)不需要任何参数，而且不能与 DISTINCT 一起使用。COUNT(*)不需要 expression 参数，因为根据定义，该函数不使用有关任何特定列的信息。COUNT(*)返回指定表中行数而不删除副本。它对各行分别计数。包括包含空值的行。

☑ 返回类型：int 类型。

【例 6.06】　使用 SELECT 语句显示学生信息，并使用 COUNT 函数统计所有学生的性别，然后使用 AS 语句，将 Sex 重命名为“人数”，最后显示查询结果，SQL 语句及运行结果如图 6.6 所示。（**实例位置：资源包\源码\06\6.06**）

SQL 语句如下：

```
USE db_2014
SELECT * FROM Student
SELECT Sex,COUNT (Sex) AS  人数  FROM Student
GROUP BY Sex
```

【例 6.07】　查询 Student 表中的总人数，SQL 语句及运行结果如图 6.7 所示。（**实例位置：资源包\源码\06\6.07**）

图 6.6　使用 COUNT 函数计算男女同学的人数

图 6.7　使用 COUNT 函数计算学生的总人数

SQL 语句如下：

```
USE db_2014
SELECT * FROM Student
SELECT COUNT (*) AS  总人数  FROM Student
```

6.1.7 DISTINCT（取不重复记录）函数

DISTINCT 函数，对指定的集求值，删除该集中的重复元组，然后返回结果集。

语法格式如下：

```
Distinct(Set_Expression)
```

参数 Set_Expression 表示返回集的有效多维表达式（MDX）。



说明

如果 DISTINCT 函数在指定的集中找到了重复的元组，则此函数只保留重复元组的第一个实例，同时保留该集原来的顺序。

【例 6.08】 使用 DISTINCT 函数查询 Course 表中不重复的课程信息，SQL 语句及运行结果如图 6.8 所示。（**实例位置：资源包\源码\06\6.08**）

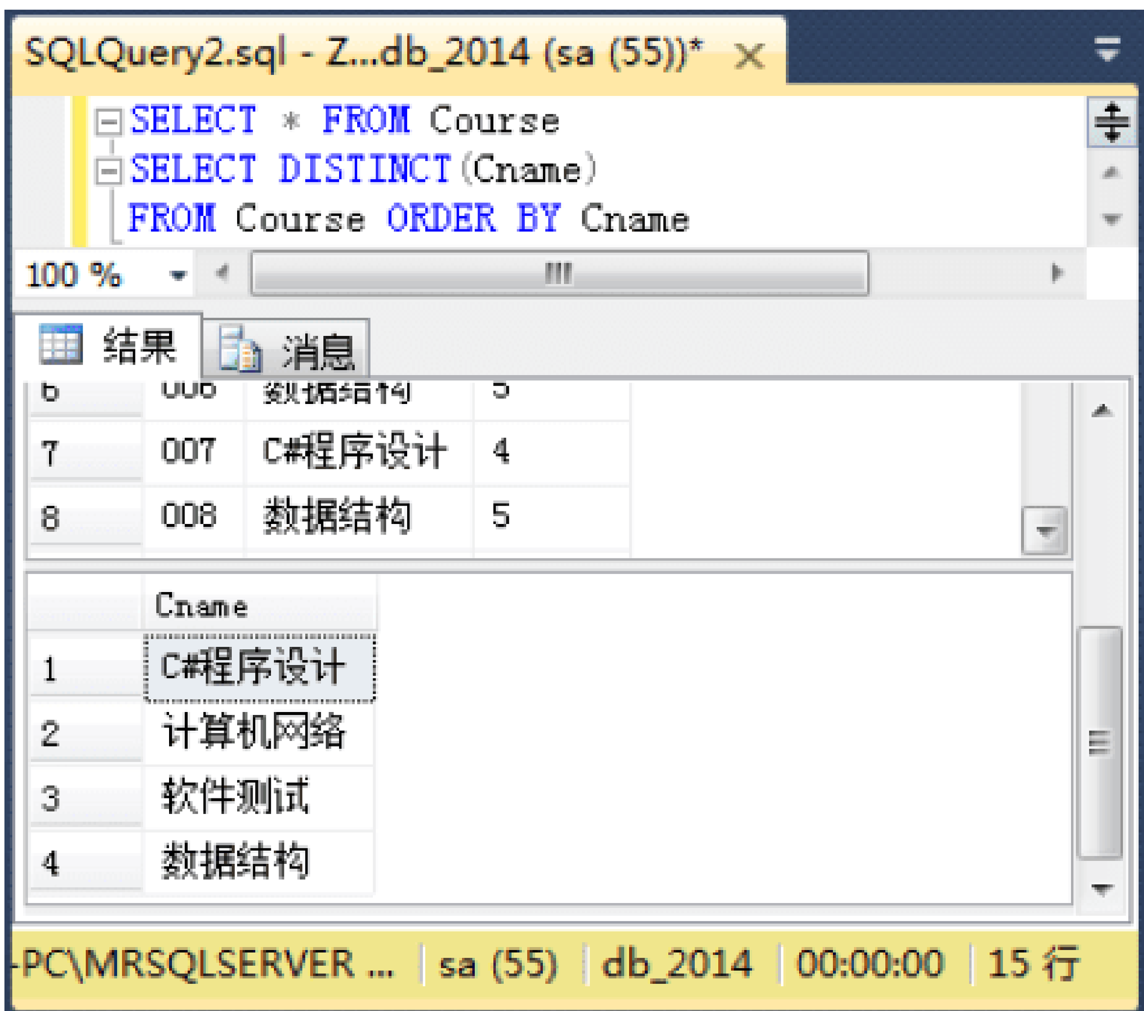


图 6.8 查询 Course 表中不重复的课程信息

SQL 语句如下：

```
SELECT * FROM Course
SELECT DISTINCT(Cname)
FROM Course ORDER BY Cname
```

6.1.8 查询重复记录

查询数据表中的重复记录，可以借助 HAVING 子句实现，该子句用来指定组或聚合的搜索条件。HAVING 子句只能与 SELECT 语句一起使用，而且它通常在 GROUP BY 子句中使用。

HAVING 子句语法格式如下：

```
[HAVING <search condition>]
```

参数 search condition 用来指定组或聚合应满足的搜索条件。

【例 6.09】　使用 HAVING 子句为组指定条件，当同种课程的记录大于等于一条时，显示此课程的名称及重复数量，SQL 语句及运行结果如图 6.9 所示。（**实例位置：资源包\源码\06\6.09**）

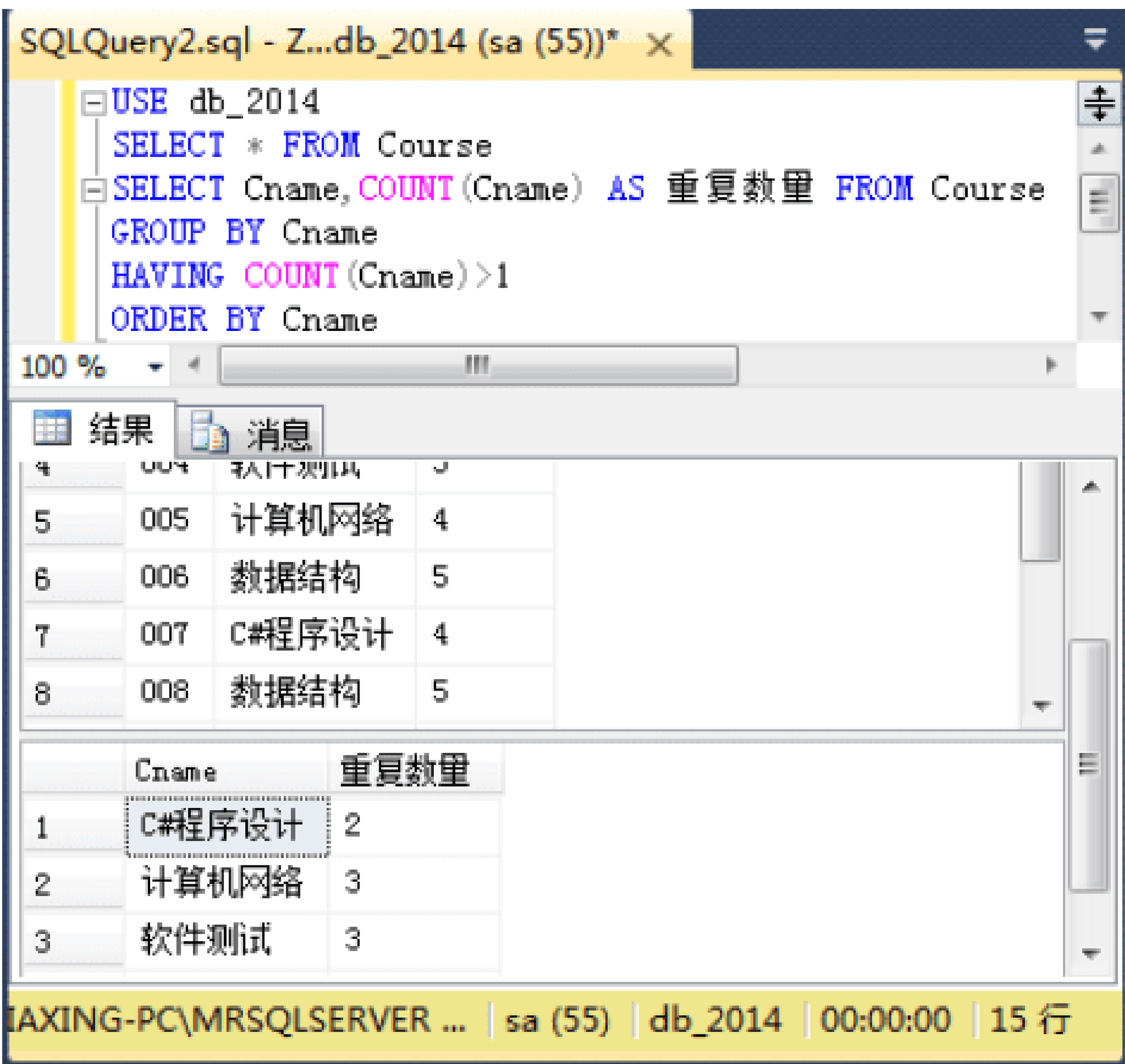


图 6.9　查询重复的课程及重复数量

SQL 语句如下：

```
USE db_2014
SELECT * FROM Course
SELECT Cname,COUNT(Cname) AS 重复数量 FROM Course
GROUP BY Cname
HAVING COUNT(Cname)>1
ORDER BY Cname
```



6.2　数 学 函 数

数学函数能够对数字表达式进行数学运算，并能够将结果返回给用户。默认情况下，传递给数学函数的数字将被解释为双精度浮点数。

6.2.1　数学函数概述

数学函数可以对数据类型为整型（integer）、实型（real）、浮点型（float）、货币型（money）和 smallmoney 的列进行操作。它的返回值是 6 位小数，如果使用出错，则返回 NULL 值并显示提示信息，通常该函数可以用在 SQL 语句的表达式中。常用的数学函数及说明如表 6.3 所示。

表 6.3　常用的数学函数及说明

函 数 名 称	说　　明
ABS	返回指定数字表达式的绝对值
COS	返回指定的表达式中指定弧度的三角余弦值
COT	返回指定的表达式中指定弧度的三角余切值
PI	返回值为圆周率
POWER	将指定的表达式乘指定次方
RAND	返回 0～1 之间的随机浮点数
ROUND	将数字表达式四舍五入为指定的长度或精度
SIGN	返回指定表达式的零（0）、正号（+1）或负号（-1）
SIN	返回指定的表达式中指定弧度的三角正弦值
SQUARE	返回指定表达式的平方
SQRT	返回指定表达式的平方根
TAN	返回指定的表达式中指定弧度的三角正切值


注意

算术函数（如 ABS、CEILING、DEGREES、FLOOR、POWER、RADIANS 和 SIGN）返回与输入值具有相同数据类型的值。三角函数和其他函数（包括 EXP、LOG、LOG10、SQUARE 和 SQRT）将输入值转换为 float 并返回 float 值。

6.2.2　ABS（绝对值）函数

ABS 函数返回数值表达式的绝对值。

语法格式如下：

```
ABS(numeric_expression)
```

参数说明如下。

☑　numeric_expression：是有符号或无符号的数值表达式。

☑　结果类型：提交给函数的数值表达式的数据类型。


说明

如果该参数为空，则 ABS 返回的结果为空。

【例 6.10】　使用 ABS 函数求指定表达式的绝对值，SQL 语句及运行结果如图 6.10 所示。（**实例位置：资源包\源码\06\6.10**）

SQL 语句如下：

```
SELECT ABS(1.0) AS "1.0 的绝对值",
ABS(0.0) AS "0.0 的绝对值",
ABS(-1.0) AS "-1.0 的绝对值"
```

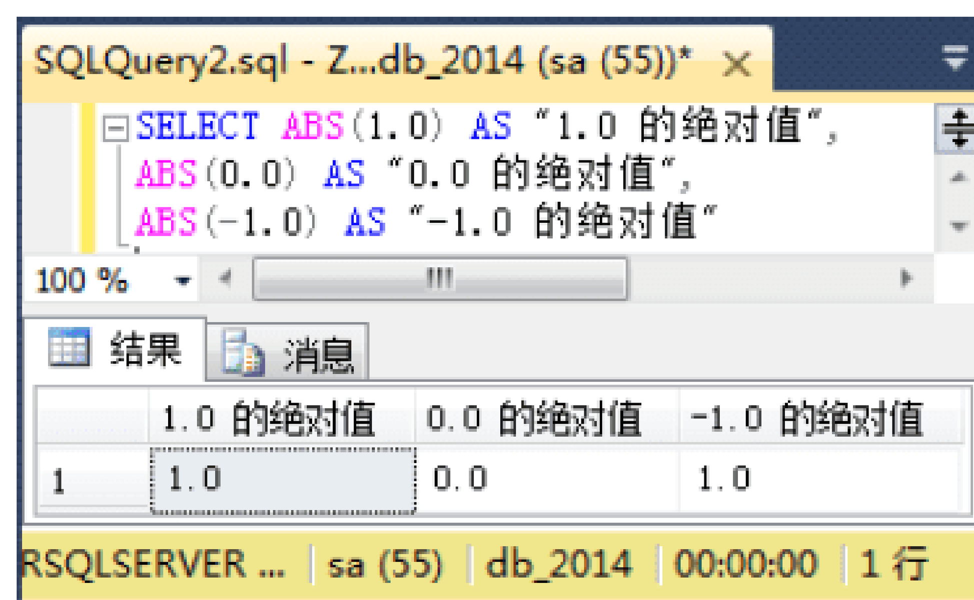


图 6.10　指定表达式的绝对值

6.2.3　PI（圆周率）函数

PI 函数返回 PI 的常量值。

语法格式如下：

```
PI( )
```

返回类型：float 型。

【例 6.11】　使用 PI 函数返回指定 PI 的值，SQL 语句及运行结果如图 6.11 所示。（**实例位置：资源包\源码\06\6.11**）

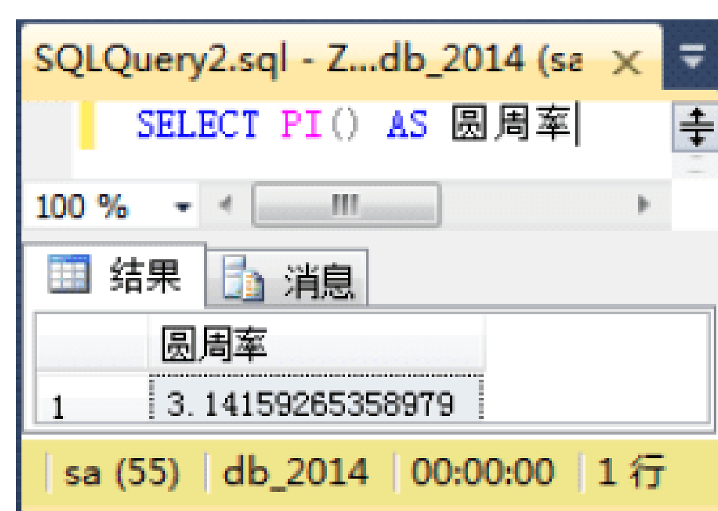


图 6.11　返回 PI 的值

SQL 语句如下：

```
SELECT PI() AS 圆周率
```

6.2.4　POWER（乘方）函数

POWER 函数返回对数值表达式进行幂运算的结果。power 参数的计算结果必须为整数。

语法格式如下：

```
POWER(numeric_expression,power)
```

参数说明如下。

☑　numeric_expression：有效的数值表达式。

☑　power：有效的数值表达式。

【例 6.12】　使用 POWER 函数分别求 2、3、4 的乘方的结果，SQL 语句及运行结果如图 6.12 所

示。（**实例位置：资源包\源码\06\6.12**）

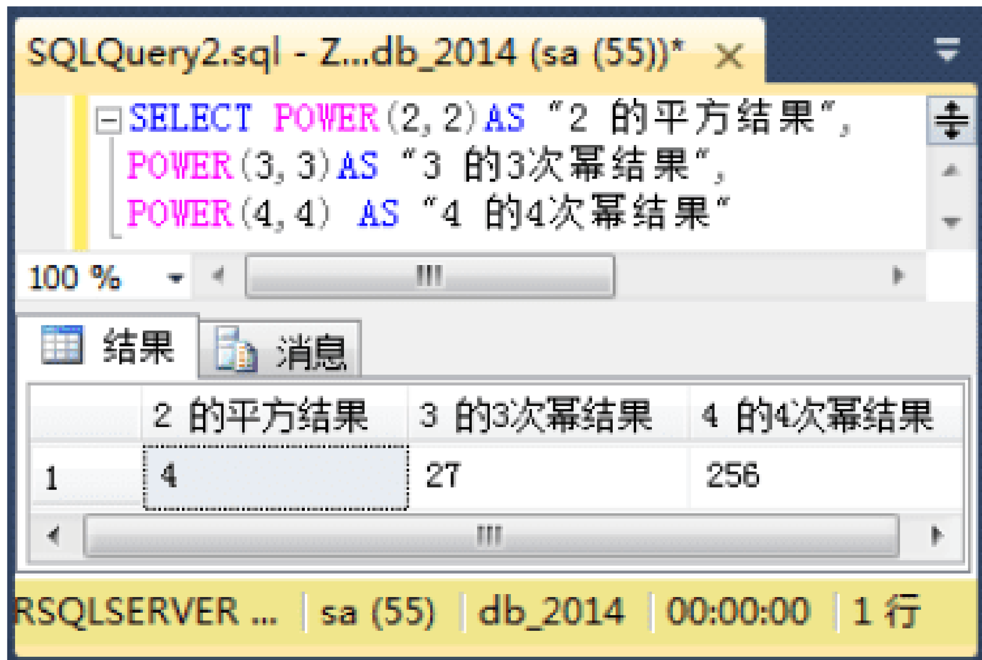


图 6.12　计算指定数的乘方

SQL 语句如下：

```
SELECT POWER(2,2) AS "2 的平方结果",
POWER(3,3) AS "3 的 3 次幂结果",
POWER(4,4) AS "4 的 4 次幂结果"
```

6.2.5　RAND（随机浮点数）函数

RAND 函数返回 0～1 之间的随机 float 值。

语法格式如下：

```
RAND([seed])
```

参数说明如下。

- ☑ seed：提供种子值的整数表达式（tinyint、smallint 或 int）。如果未指定 seed，则 SQL Server 数据库引擎随机分配种子值。对于指定的种子值，返回的结果始终相同。
- ☑ 返回类型：float 类型。

注意

使用同一个种子值重复调用 RAND()会返回相同的结果。

【例 6.13】 使用同一种子值调用 RAND 函数，返回相同的数字序列，SQL 语句及运行结果如图 6.13 所示。（**实例位置：资源包\源码\06\6.13**）



图 6.13　使用同一种子值调用 RAND 函数

SQL 语句如下：

```
SELECT RAND(100), RAND(100), RAND()
```

【例 6.14】　使用 RAND 函数生成的 3 个不同的随机数，SQL 语句及运行结果如图 6.14 所示。（**实例位置：资源包\源码\06\6.14**）

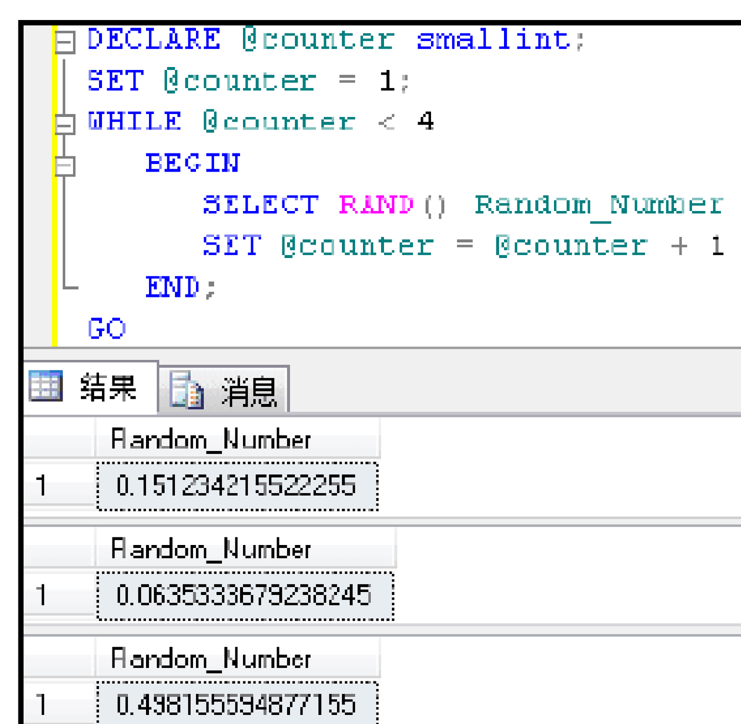


图 6.14　生成的 3 个不同的随机数

SQL 语句如下：

```
DECLARE @counter smallint;
SET @counter = 1;
WHILE @counter < 4
    BEGIN
        SELECT RAND() Random_Number
        SET @counter = @counter + 1
    END;
GO
```

6.2.6　ROUND（四舍五入）函数

ROUND 函数返回一个数值，舍入到指定的长度或精度。

语法格式如下：

```
ROUND(numeric_expression, length [,function])
```

参数说明如下。

- ☑ numeric_expression：精确数值或近似数值数据类别（bit 数据类型除外）的表达式。
- ☑ length：numeric_expression 的舍入精度。length 必须是 tinyint、smallint 或 int 类型的表达式。如果 length 为正数，则将 numeric_expression 舍入到 length 指定的小数位数。如果 length 为负数，则将 numeric_expression 小数点左边部分舍入到 length 指定的长度。
- ☑ function：要执行的操作的类型。function 必须为 tinyint、smallint 或 int。如果省略 function 或其值为 0（默认值），则将舍入 numeric_expression。如果指定了 0 以外的值，则将截断 numeric_expression。
- ☑ 返回类型：返回与 numeric_expression 相同的类型。

【例 6.15】 使用 ROUND 函数计算指定表达式的值，SQL 语句及运行结果如图 6.15 所示。（**实例位置：资源包\源码\06\6.15**）

图 6.15 使用 ROUND 函数计算表达式

SQL 语句如下：

```
SELECT ROUND(123.9994, 3), ROUND(123.9995, 3)
```

6.2.7 SQUARE（平方）函数和 SQRT（平方根）函数

1. SQUARE（平方）函数

SQUARE 函数返回数值表达式的平方。

语法格式如下：

```
SQUARE(numeric_expression)
```

参数 numeric_expression 表示任意数值数据类型的数值表达式。

【例 6.16】 使用 SQUARE 函数计算指定表达式的值，SQL 语句及运行结果如图 6.16 所示。（**实例位置：资源包\源码\06\6.16**）

图 6.16 使用 SQUARE 函数计算表达式

SQL 语句如下：

```
SELECT SQUARE(4) AS "4 的平方"
```

2. SQRT（平方根）函数

SQRT 函数返回数值表达式的平方根。

语法格式如下：

```
SQRT(numeric_expression)
```

参数 numeric_expression 表示任意数值数据类型的数值表达式。

【例 6.17】　使用 SQRT 函数计算指定表达式的值，SQL 语句及运行结果如图 6.17 所示。（**实例位置：资源包\源码\06\6.17**）

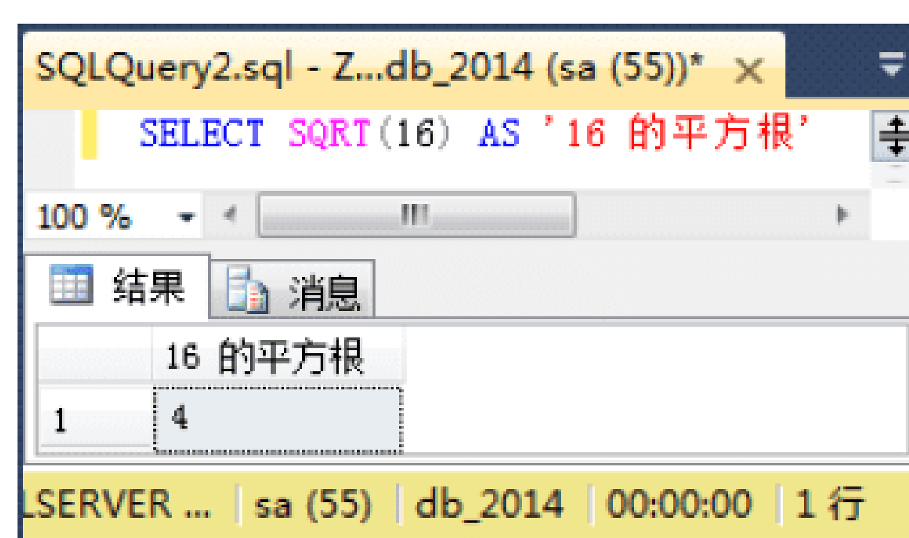


图 6.17　使用 SQRT 函数计算表达式

SQL 语句如下：

```
SELECT SQRT(16) AS '16 的平方根'
```

【例 6.18】　使用 SQRT 函数返回 1.00～10.00 之间的数字平方根。（**实例位置：资源包\源码\06\6.18**）

SQL 语句如下：

```
DECLARE @mysqrt float
SET @mysqrt = 1.00
WHILE @mysqrt < 10.00
BEGIN
  SELECT SQRT(@mysqrt)
  SELECT @mysqrt = @mysqrt + 1
END
```

程序运行结果如表 6.4 所示。

表 6.4　结果集

数　字	平　方　根	数　字	平　方　根
1.00	1.0	6.00	2.44948974278318
2.00	1.4142135623731	6.00	2.64575131106459
3.00	1.73205080756888	8.00	2.82842712474619
4.00	2.0	9.00	3.0
5.00	2.23606797749979		

6.2.8　三角函数

三角函数包括 COS、COT、SIN 以及 TAN 函数，分别表示为三角余弦值、三角余切值、三角正弦值和三角正切值，下面分别对这几种三角函数进行详细讲解。

1．COS 函数

COS 函数返回指定表达式中以弧度表示的指定角的三角余弦。

语法格式如下：

```
COS(float_expression)
```

参数说明如下。

☑ float_expression：float 类型的表达式。

☑ 返回类型：float 类型。

【例 6.19】 使用 COS 函数返回指定表达式的余弦值，SQL 语句及运行结果如图 6.18 所示。（**实例位置：资源包\源码\06\6.19**）

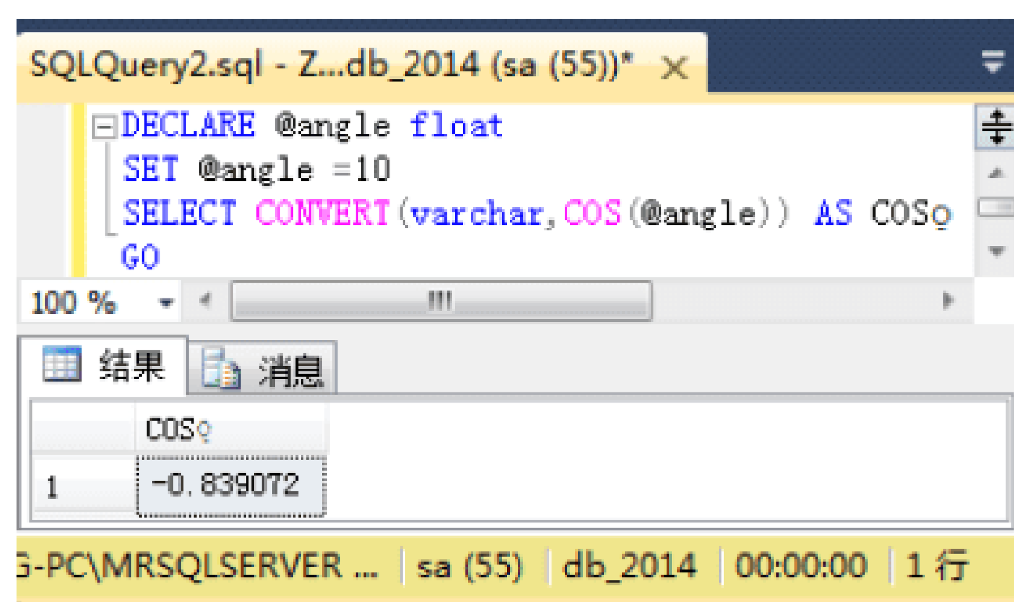


图 6.18 方法指定表达式的余弦值

SQL 语句如下：

```
DECLARE @angle float
SET @angle =10
SELECT CONVERT(varchar,COS(@angle)) AS COS值
GO
```

2．COT 函数

COT 函数返回指定表达式中以弧度表示的指定角的三角余切值。

语法格式如下：

```
COT(float_expression)
```

参数说明如下。

☑ float_expression：float 类型或能够隐式转换为 float 类型的表达式。

☑ 返回类型：float 类型。

【例 6.20】 使用 COT 函数返回指定表达式的余切值，SQL 语句及运行结果如图 6.19 所示。（**实例位置：资源包\源码\06\6.20**）

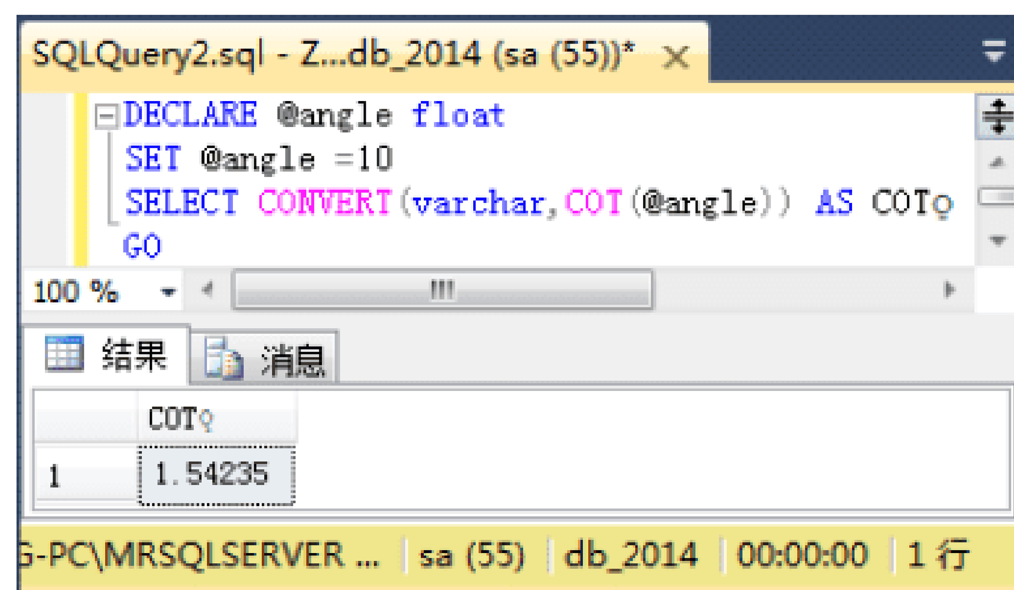


图 6.19 返回指定表达式的余切值

SQL 语句如下：

```
DECLARE @angle float
SET @angle =10
SELECT CONVERT(varchar,COT(@angle)) AS COT
GO
```

3．SIN 函数

SIN 函数返回指定表达式中以弧度表示的指定角的三角正弦值。

语法格式如下：

```
SIN(float_expression)
```

参数说明如下。

☑　float_expression：float 类型或能够隐式转换为 float 类型的表达式。

☑　返回类型：float 类型。

【例 6.21】　使用 SIN 函数返回指定表达式的正弦值，SQL 语句及运行结果如图 6.20 所示。（**实例位置：资源包\源码\06\6.21**）

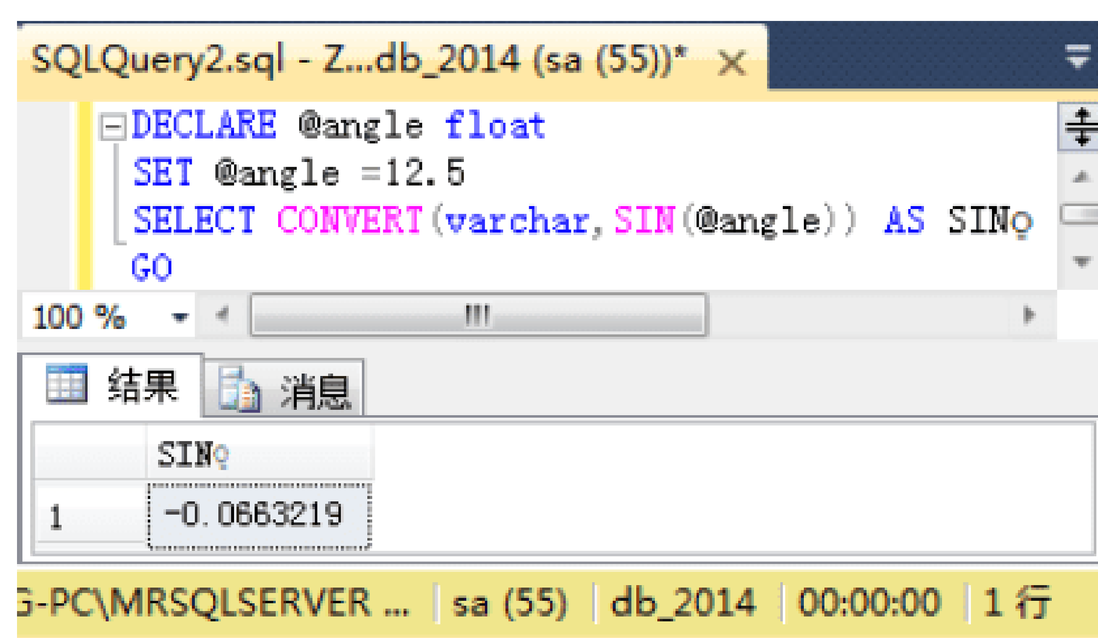


图 6.20　返回指定表达式的正弦值

SQL 语句如下：

```
DECLARE @angle float
SET @angle =12.5
SELECT CONVERT(varchar,SIN(@angle)) AS SIN
GO
```

4．TAN 函数

TAN 函数返回指定表达式中以弧度表示的指定角的三角正切值。

语法格式如下：

```
TAN(float_expression)
```

参数说明如下。

☑　float_expression：float 类型或可隐式转换为 float 类型的表达式。

☑　返回类型：float 类型。

【例 6.22】 使用 TAN 函数返回指定表达式的正切值，SQL 语句及运行结果如图 6.21 所示。（**实例位置：资源包\源码\06\6.22**）

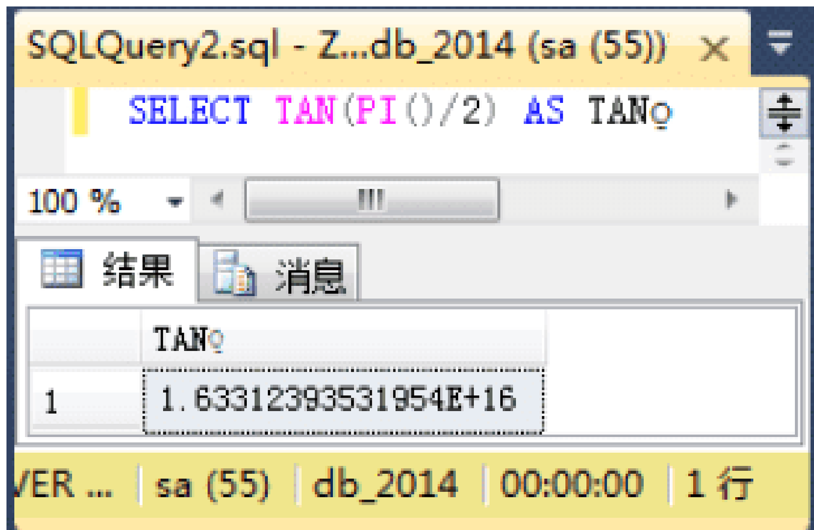


图 6.21 返回指定表达式的正切值

SQL 语句如下：

```
SELECT TAN(PI()/2) AS TAN
```



6.3 字符串函数

字符串函数对 N 进制数据、字符串和表达式执行不同的运算，如返回字符串的起始位置，返回字符串的个数等。本节向读者介绍 SQL Server 中常用的字符串函数。

6.3.1 字符串函数概述

字符串函数作用于 char、varchar、binary 和 varbinary 数据类型以及可以隐式转换为 char 或 varchar 的数据类型。通常字符串函数可以用在 SQL 语句的表达式中。常用的字符串函数及说明如表 6.5 所示。

表 6.5 常用的字符串函数及说明

函 数 名 称	说 明
ASCII	返回字符表达式最左端字符的 ASCII 代码值
CHARINDEX	返回字符串中指定表达式的起始位置
LEFT	从左边开始，取得字符串左边指定个数的字符
LEN	返回指定字符串的字符（而不是字节）个数
REPLACE	将指定的字符串替换为另一指定的字符串
REVERSE	返回字符表达式的反转
RIGHT	从右边开始，取得字符串右边指定个数的字符
STR	返回由数字数据转换来的字符数据
SUBSTRING	返回指定个数的字符

6.3.2 ASCII（获取 ASCII 码）函数

ASCII 函数返回字符表达式中最左侧的字符的 ASCII 代码值。

语法格式如下：

```
ASCII(character_expression)
```

参数说明如下。

☑　character_expression：char 或 varchar 类型的表达式。

☑　返回类型：int 类型。



说明

ASCII 码共有 127 个，其中 Microsoft Windows 不支持 1～7、11～12 和 14～31 之间的字符。值 8、9、10 和 13 分别转换为退格、制表、换行和回车字符。它们并没有特定的图形显示，但会依不同的应用程序而对文本显示有不同的影响。

ASCII 码值对照表如表 6.6 所示。

表 6.6　ASCII 码值对照表

ASCII 码	按　　键	ASCII 码	按　　键	ASCII 码	按　　键	ASCII 码	按　　键
32	[space]	64	@	96	`	115	s
33	!	65	A	97	A	116	t
34	"	66	B	98	B	117	u
35	#	67	C	99	C	118	v
36	$	68	D	100	D	119	w
37	%	69	E	101	E	120	x
38	&	70	F	102	F	121	y
39	'	71	G	103	G	122	z
40	(	72	H	104	H	123	{
41	)	73	I	105	I	124	|
42	*	74	J	106	j	125	}
43	+	75	K	107	k	126	~
44	,	76	L	108	l		
45	-	77	M	109	m		
46	.	78	N	110	n		
47	/	79	O	111	o		
48	0	80	P	112	p		
49	1	81	Q	113	q		
50	2	82	R	114	r		

【例 6.23】　使用 ASCII 函数返回 NXT 的 ASCII 代码值，SQL 语句及运行结果如图 6.22 所示。（**实例位置：资源包\源码\06\6.23**）

SQL 语句如下：

```
DECLARE @position int, @string char(3)
SET @position = 1
```

```
SET @string = 'NXT'
WHILE @position <= DATALENGTH(@string)
    BEGIN
    SELECT ASCII(SUBSTRING(@string, @position, 1)) AS ASCII 值,
        CHAR(ASCII(SUBSTRING(@string, @position, 1))) AS 字符
    SET @position = @position + 1
    END
```

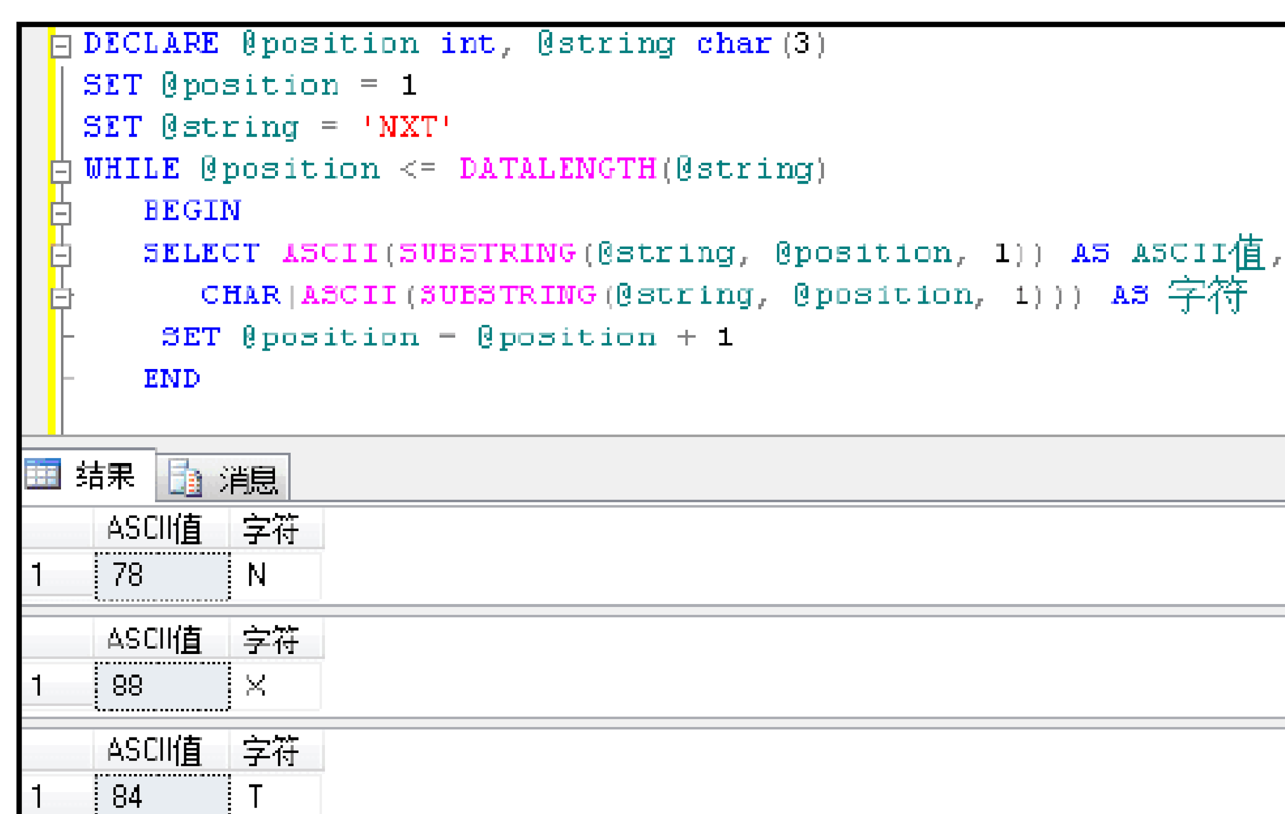


图 6.22　返回指定表达式的 ASCII 值

6.3.3　CHARINDEX（返回字符串的起始位置）函数

CHARINDEX 函数返回字符串中指定表达式的起始位置（如果找到）。搜索的起始位置为 start_location。

语法格式如下：

```
CHARINDEX(expression1,expression2 [, start_location])
```

参数说明如下。

☑ expression1：包含要查找的序列的字符表达式。expression1 最大长度限制为 8000 个字符。

☑ expression2：要搜索的字符表达式。

☑ start_location：表示搜索起始位置的整数或 bigint 表达式。如果未指定 start_location，或者 start_location 为负数或 0，则将从 expression2 的开头开始搜索。

☑ 返回类型：如果 expression2 的数据类型为 varchar(max)、nvarchar(max) 或 varbinary(max)，则为 bigint，否则为 int。

【例 6.24】 使用 CHARINDEX 函数返回指定字符串的起始位置，SQL 语句及运行结果如图 6.23 所示。（**实例位置：资源包\源码\06\6.24**）

SQL 语句如下：

```
USE db_2014
SELECT * FROM Course
SELECT CHARINDEX('设计',Cname) AS "起始位置" FROM Course
WHERE Cno = '003'
```

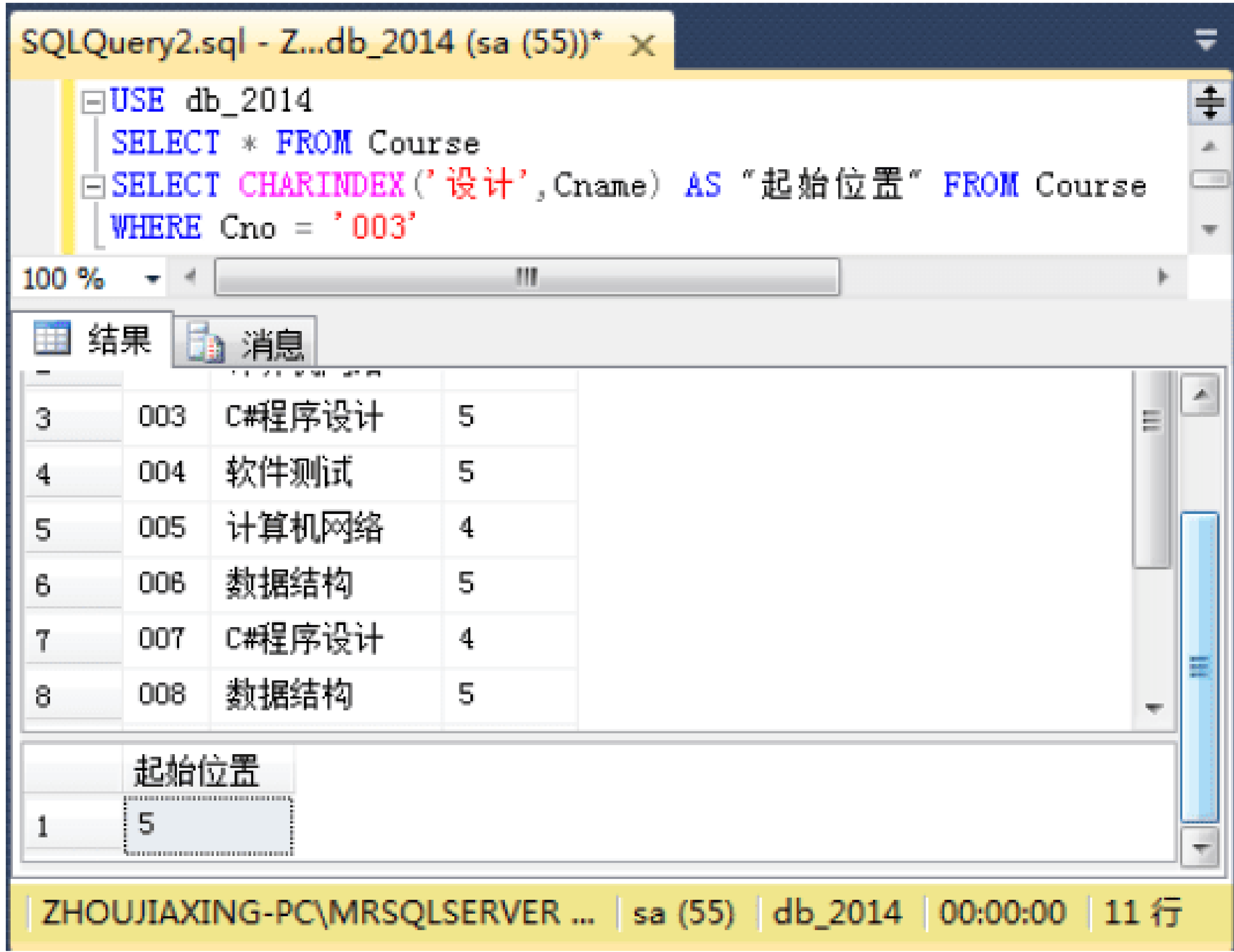


图 6.23　返回指定字符串的起始位置

6.3.4　LEFT（取左边指定个数的字符）函数

LEFT 函数返回字符串中从左边开始指定个数的字符。

语法格式如下：

```
LEFT(character_expression, integer_expression)
```

参数说明如下。

☑ character_expression：字符或二进制数据表达式。character_expression 可以是常量、变量或列。character_expression 可以是任何能够隐式转换为 varchar 或 nvarchar 的数据类型，但 text 或 ntext 除外。否则，请使用 CAST 函数对 character_expression 进行显式转换。

☑ integer_expression：正整数，指定 character_expression 将返回的字符数。如果 integer_expression 为负，则将返回错误。如果 integer_expression 的数据类型为 bigint 且包含一个较大值，character_expression 必须是大型数据类型，如 varchar(max)。

返回类型如下。

☑ 当 character_expression 为非 Unicode 字符数据类型时，返回 varchar。

☑ 当 character_expression 为 Unicode 字符数据类型时，返回 nvarchar。

【例 6.25】　使用 LEFT 函数返回指定字符串的最左边 4 个字符，SQL 语句及运行结果如图 6.24 所示。（**实例位置：资源包\源码\06\6.25**）

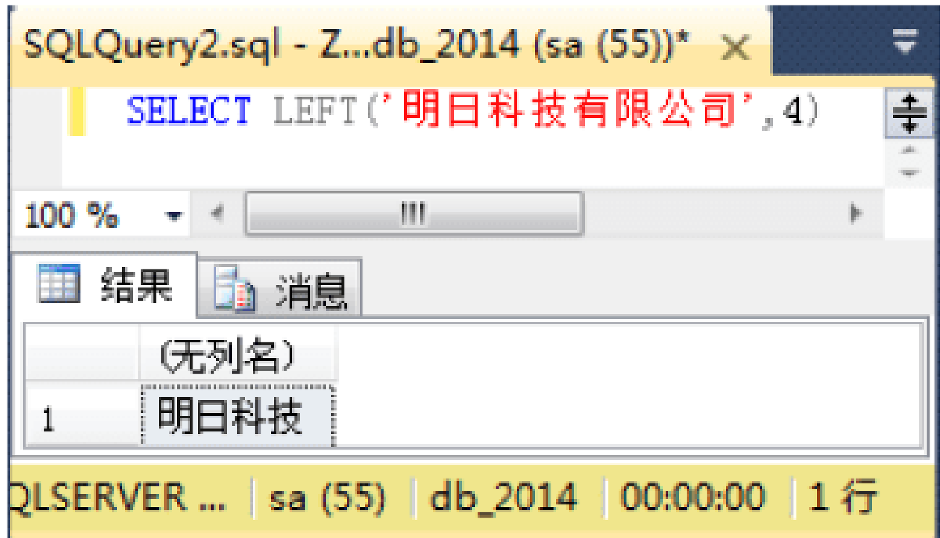


图 6.24　返回指定字符串中的字符

SQL 语句如下：

```
SELECT LEFT('明日科技有限公司',4)
```

【例 6.26】 使用 LEFT 函数查询 Student 表中的姓氏（姓氏是姓名的第一位）并计算出每个姓氏的数量，SQL 语句及运行结果如图 6.25 所示。（**实例位置：资源包\源码\06\6.26**）

图 6.25　查询 Student 表中的姓氏

SQL 语句如下：

```
USE db_2014
SELECT Sno,Sname FROM Student
SELECT LEFT(Sname,1) AS '姓氏', COUNT(LEFT(Sname,1)) AS '数量'
FROM Student Group BY LEFT(Sname,1)
```

6.3.5　RIGHT（取右边指定个数的字符）函数

RIGHT 函数返回字符表达式中从起始位置（从右端开始）到指定字符位置（从右端开始计数）的部分。

语法格式如下：

```
RIGHT(character_expression,integer_expression)
```

参数说明如下。

☑　character_expression：从中提取字符的字符表达式。

☑　number：指示返回字符数的整数表达式。

【例 6.27】 使用 RIGHT 函数查询 Student 表中编号的后 3 位，SQL 语句及运行结果如图 6.26 所

示。（**实例位置：资源包\源码\06\6.27**）

SQLQuery2.sql - Z...db_2014 (sa (55))*

```
USE db_2014
SELECT Sno, Sname, Sex FROM Student
SELECT RIGHT(Sno, 4) AS '编号' , Sname, Sex
FROM Student
```

100 %

结果　消息

	Sno	Sname	Sex
1	201109001	李羽凡	男
2	201109002	王都都	女
3	201109003	聂乐乐	女
4	201109004	张东健	男
5	201109005	王子	男
6	201109006	邢星	女
7	201109008	赵雪	女

	编号	Sname	Sex
1	001	李羽凡	男
2	002	王都都	女
3	003	聂乐乐	女
4	004	张东健	男
5	005	王子	男

图 6.26　查询 Student 表中的编号后 3 位

SQL 语句如下：

```
USE db_2014
SELECT Sno,Sname,Sex FROM Student
SELECT RIGHT(Sno,4) AS '编号',Sname,Sex
FROM Student
```

6.3.6　LEN（返回字符个数）函数

LEN 函数返回字符表达式中的字符数。如果字符串中包含前导空格和尾随空格，则函数会将它们包含在计数内。LEN 对相同的单字节和双字节字符串返回相同的值。

语法格式如下：

```
LEN(character_expression)
```

参数 character_expression 表示要处理的表达式。

【例 6.28】　使用 LEN 函数计算指定字符的个数，SQL 语句及运行结果如图 6.27 所示。（**实例位置：资源包\源码\06\6.28**）

SQL 语句如下：

```
SELECT LEN('ABCDE') AS "字符个数"
SELECT LEN('NIEXITING') AS "字符个数"
SELECT LEN('吉林省明日科技有限公司') AS "字符个数"
```

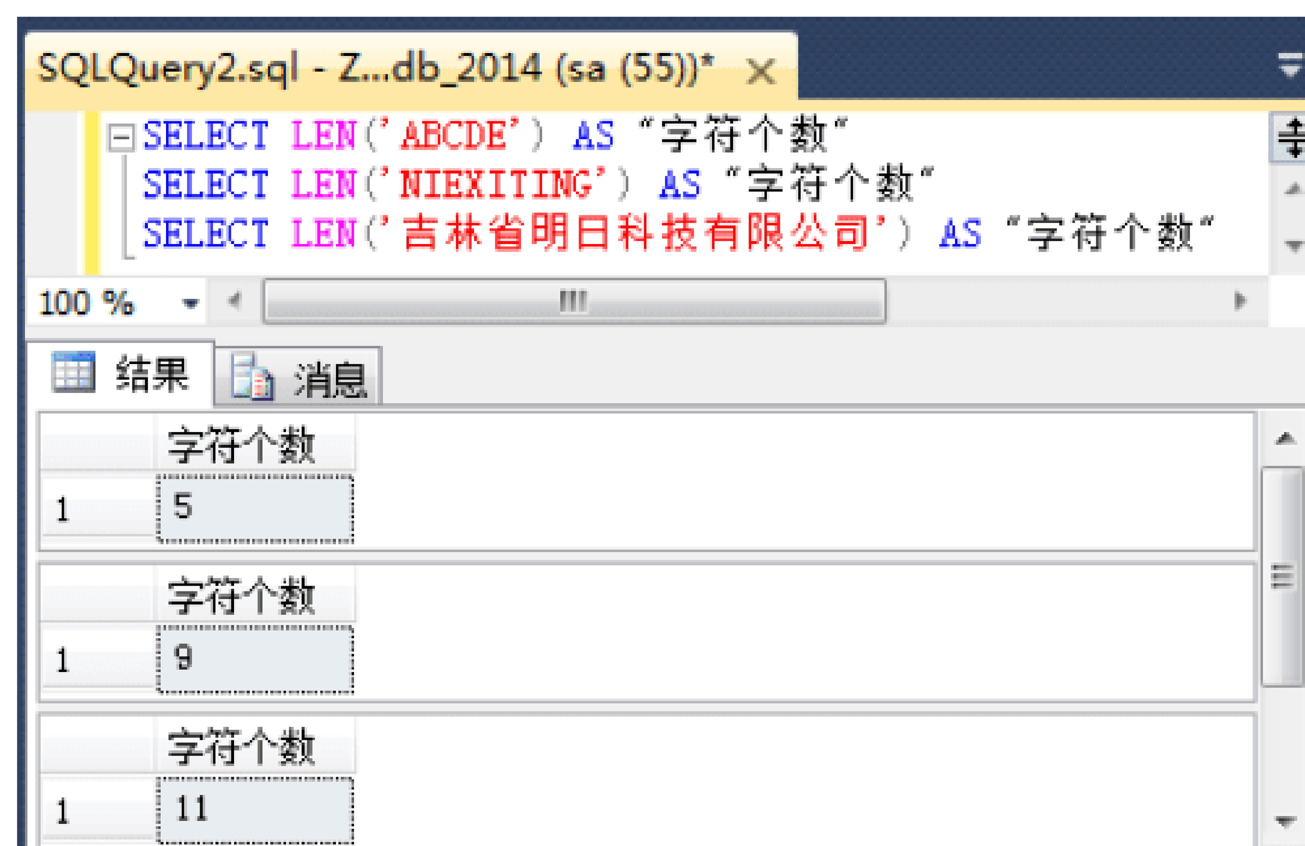


图 6.27　指定字符的个数

6.3.7　REPLACE（替换字符串）函数

REPLACE 函数将表达式中的一个字符串替换为另一个字符串或空字符串后，返回一个字符表达式。

语法格式如下：

```
REPLACE(character_expression,searchstring,replacementstring)
```

参数说明如下。

☑　character_expression：函数要搜索的有效字符表达式。

☑　searchstring：函数尝试定位的有效字符表达式。

☑　replacementstring：用作替换表达式的有效字符表达式。

【例 6.29】　使用 REPLACE 函数替换指定的字符，SQL 语句及运行结果如图 6.28 所示。（**实例位置：资源包\源码\06\6.29**）

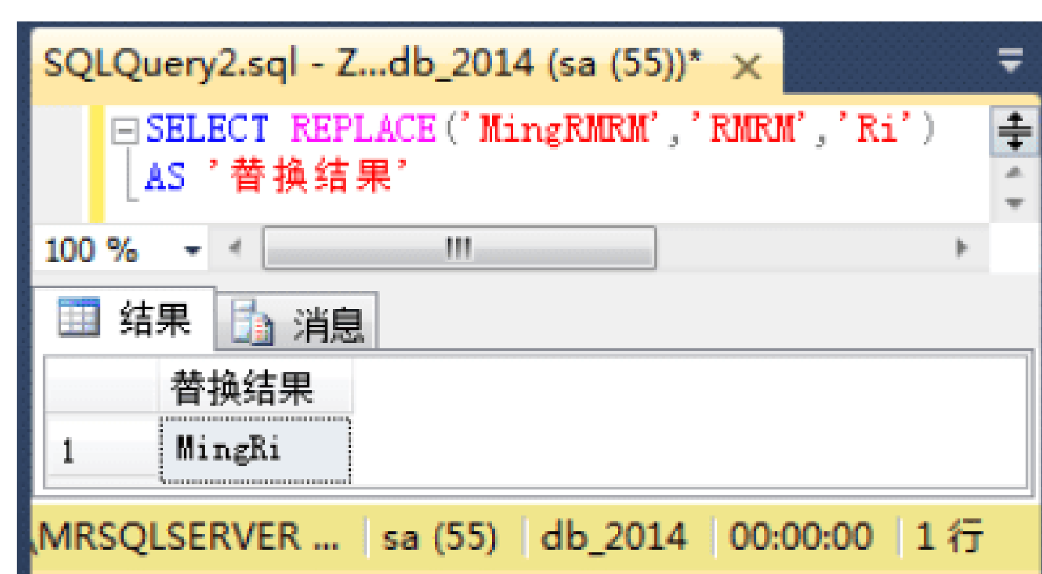


图 6.28　替换指定的字符

SQL 语句如下：

```
SELECT REPLACE('MingRMRM','RMRM','Ri')
AS '替换结果'
```

6.3.8　REVERSE（返回字符表达式的反转）函数

REVERSE 函数按相反顺序返回字符表达式。

语法格式如下：

```
REVERSE(character_expression)
```

参数 character_expression 表示要反转的字符表达式。

【例 6.30】　使用 REVERSE 函数反转指定的字符，SQL 语句及运行结果如图 6.29 所示。（**实例位置：资源包\源码\06\6.30**）



图 6.29　反转指定的字符

SQL 语句如下：

```
SELECT REVERSE ('irgnim')
AS '反转结果'
```

6.3.9　STR 函数

STR 函数返回由数字数据转换来的字符数据。

语法格式如下：

```
STR(float_expression [, length [, decimal]])
```

参数说明如下。

☑　float_expression：带小数点的近似数字（float）数据类型的表达式。

☑　length：总长度。它包括小数点、符号、数字以及空格。默认值为 10。

☑　decimal：小数点后的位数。decimal 必须小于或等于 16。如果 decimal 大于 16，则会截断结果，使其保持为小数点后具有 16 位。

【例 6.31】　使用 STR 函数返回以下字符数据，SQL 语句及运行结果如图 6.30 所示。（**实例位置：资源包\源码\06\6.31**）



图 6.30　使用 STR 函数转换字符串

SQL 语句如下：

```
SELECT STR(123.45) AS 'STR',
STR(123.45,5,1) AS 'STR',
STR(123.45,8,1) AS 'STR',
STR(123.45,2,2) AS 'STR'
```


注意

当表达式超出指定长度时，字符串为指定长度返回**。

6.3.10　SUBSTRING（取字符串）函数

SUBSTRING 函数返回字符表达式、二进制表达式、文本表达式或图像表达式的一部分。

语法格式如下：

```
SUBSTRING(value_expression,start_expression, length_expression)
```

参数说明如下。

☑ value_expression：是 character、binary、text、ntext 或 image 表达式。

☑ start_expression：指定返回字符的起始位置的整数或 bigint 表达式。如果 start_expression 小于 0，会生成错误并终止语句。如果 start_expression 大于值表达式中的字符数，将返回一个零长度的表达式。

☑ length_expression：是正整数或指定要返回的 value_expression 的字符数的 bigint 表达式。如果 length_expression 是负数，会生成错误并终止语句。如果 start_expression 与 length_expression 的总和大于 value_expression 中的字符数，则返回整个值表达式。

☑ 返回类型：如果 expression 是受支持的字符数据类型，则返回字符数据。如果 expression 是支持的 binary 数据类型中的一种数据类型，则返回二进制数据。返回的字符串类型与指定表达式的类型相同，表 6.7 中显示的除外。

表 6.7　返回的字符串类型与指定表达式的类型不相同

指定的表达式	返 回 类 型
char/varchar/text	varchar
nchar/nvarchar/ntext	nvarchar
binary/varbinary/image	varbinary

【例 6.32】　使用 SUBSTRING 函数，在 Sno 字段中从第 5 位开始取字符串，共 5 位，SQL 语句及运行结果如图 6.31 所示。（**实例位置：资源包\源码\06\6.32**）

SQL 语句如下：

```
SELECT Sno, SUBSTRING(Sno,5,5) AS '编号'
FROM Student
```

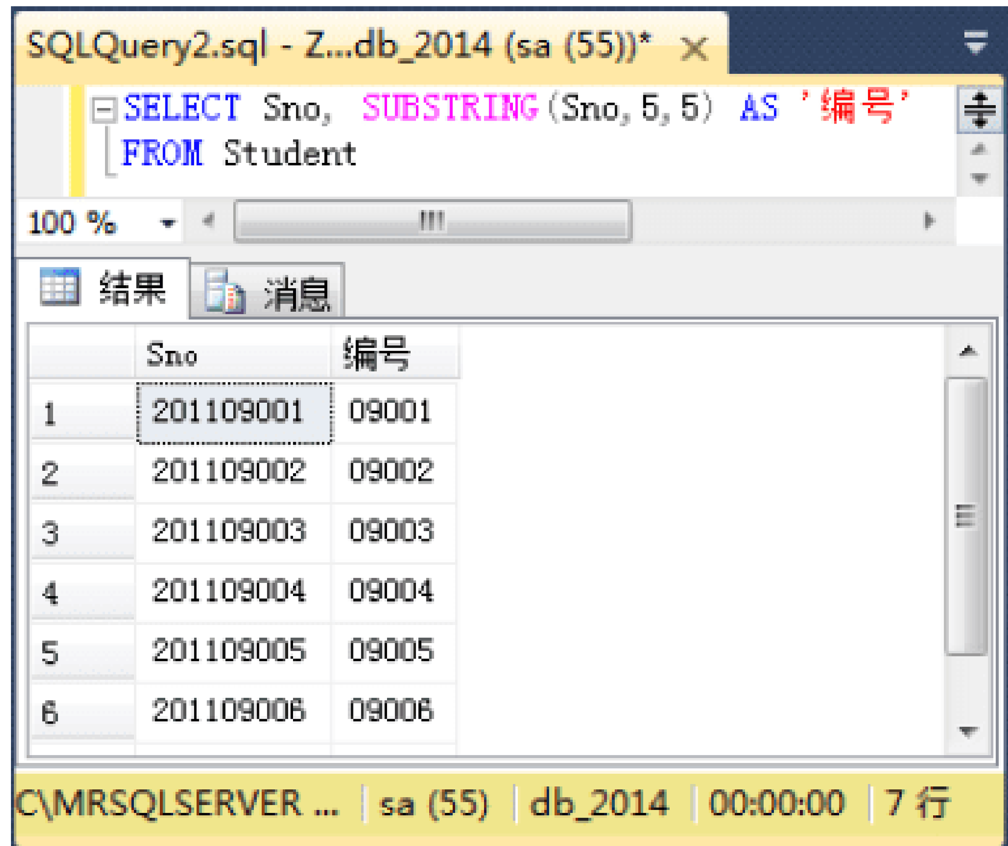


图 6.31　使用 SUBSTRING 函数取字符串

6.4　日期和时间函数



日期和时间函数主要用来显示有关日期和时间的信息。在日期和时间函数中，DAY 函数、MONTH 函数、YEAR 函数是用来获取日期和时间部分的函数。DATEDIFF 函数是用来获取日期和时间差的函数。DATEADD 函数是用来修改日期和时间值的函数。本节详细地向读者介绍这些函数。

6.4.1　日期和时间函数概述

日期和时间函数主要用来操作 datetime、smalldatetime 类型的数据，日期和时间函数执行算术运行与其他函数一样，也可以在 SQL 语句的 SELECT、WHERE 子句以及表达式中使用。常用的日期时间函数及说明如表 6.8 所示。

表 6.8　常用的日期和时间函数及说明

函 数 名 称	说　　明
DATEADD	在向指定日期加上一段时间的基础上，返回新的 datetime 值
DATEDIFF	返回跨两个指定日期的日期和时间边界数
GETDATE	返回当前系统日期和时间
DAY	返回指定日期中的天的整数
MONTH	返回指定日期中的月份的整数
YEAR	返回指定日期中的年份的整数

6.4.2　GETDATE（返回当前系统日期和时间）函数

GETDATE 函数返回系统的当前日期。GETDATE 函数不使用参数。


注意

GETDATE 函数的返回结果的长度为 29 个字符。

语法格式如下：

```
GETDATE()
```

【例 6.33】　使用 GETDATE 函数，返回当前系统日期和时间，SQL 语句及运行结果如图 6.32 所示。（**实例位置：资源包\源码\06\6.33**）

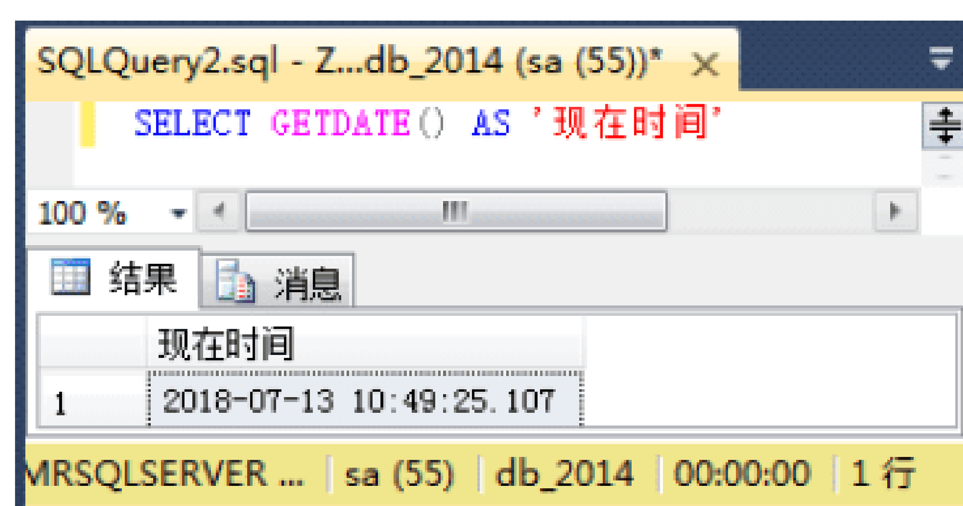


图 6.32　获取当前系统时间

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
```

6.4.3　DAY（返回指定日期的天）函数

DAY 函数返回一个整数，表示日期的“日”部分。

语法格式如下：

```
DAY(date)
```

参数 date 表示以日期格式返回有效的日期或字符串的表达式。

【例 6.34】　使用 DAY 函数，返回现有日期的“日”部分，SQL 语句及运行结果如图 6.33 所示。（**实例位置：资源包\源码\06\6.34**）

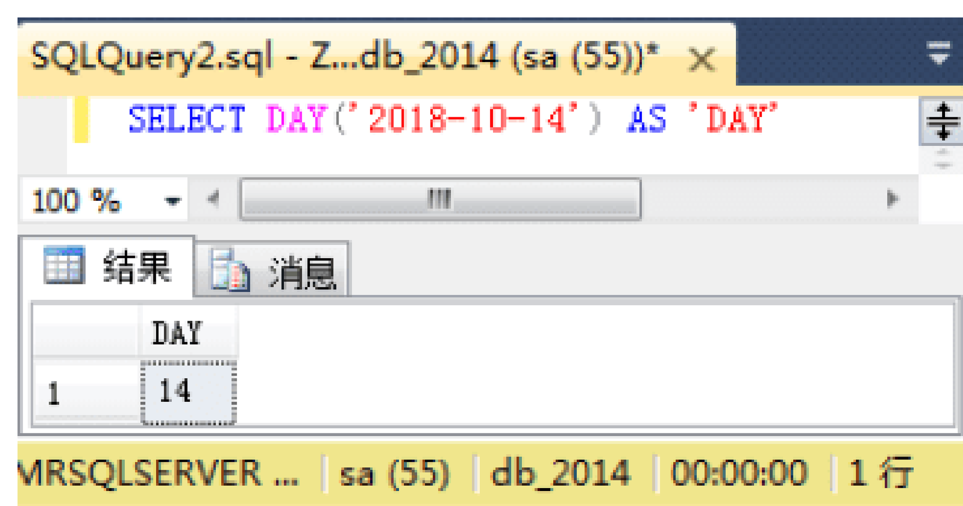


图 6.33　返回现有日期的“日”部分

SQL 语句如下：

```
SELECT DAY('2018-10-14') AS 'DAY'
```

【例 6.35】　使用 DAY 函数，返回当前日期的“日”部分，SQL 语句及运行结果如图 6.34 所示。（**实例位置：资源包\源码\06\6.35**）

SQL 语句如下：

```
SELECT DAY(GETDATE()) AS 'DAY'
```

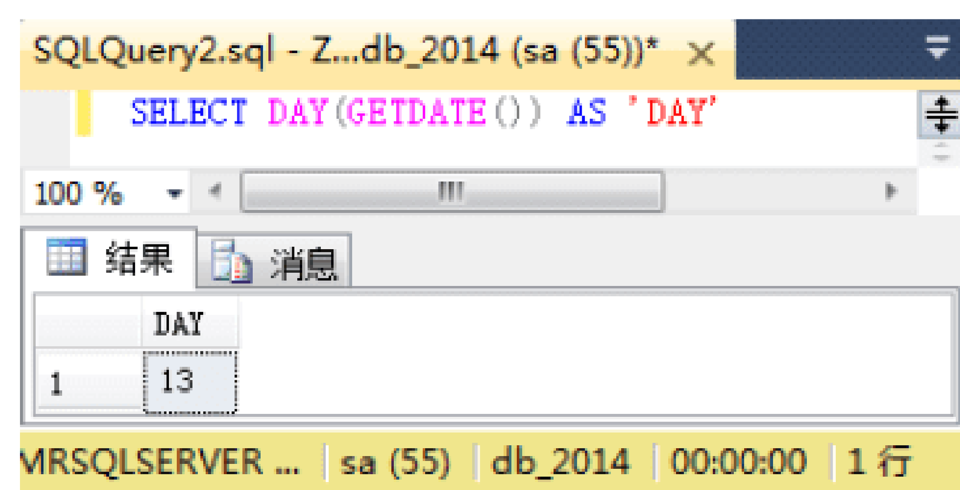


图 6.34　返回当前日期的“日”部分

6.4.4　MONTH（返回指定日期的月）函数

MONTH 函数返回一个表示日期中的“月份”部分的整数。

语法格式如下：

```
MONTH(date)
```

参数 date 表示任意日期格式的日期。

【例 6.36】　使用 MONTH 函数，返回指定日期时间的“月份”，SQL 语句及运行结果如图 6.35 所示。（**实例位置：资源包\源码\06\6.36**）

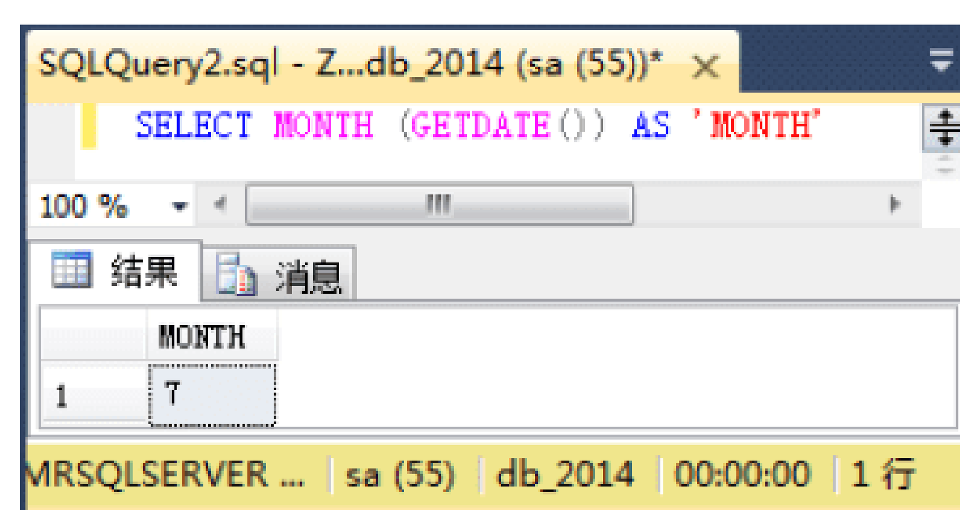


图 6.35　返回当前日期的“月份”

SQL 语句如下：

```
SELECT MONTH(GETDATE()) AS 'MONTH'
```

6.4.5　YEAR（返回指定日期的年）函数

YEAR 函数用于返回指定日期的“年份”。

语法格式如下：

```
YEAR(date)
```

参数 date 表示返回类型为 datetime 或 smalldatetime 的日期表达式。

有关 YEAR 函数使用的几点说明如下。

☑　该函数等价于 DATEPART(yy,date)。

☑　SQL Server 数据库将 0 解释为 1900 年 1 月 1 日。

☑　在使用日期函数时，其日期范围只应为 1753 年～9999 年，这是 SQL Server 系统所能识别的日期范围，否则会出现错误。

【例 6.37】 使用 YEAR 函数，返回指定日期时间的“年份”，SQL 语句及运行结果如图 6.36 所示。（**实例位置：资源包\源码\06\6.37**）

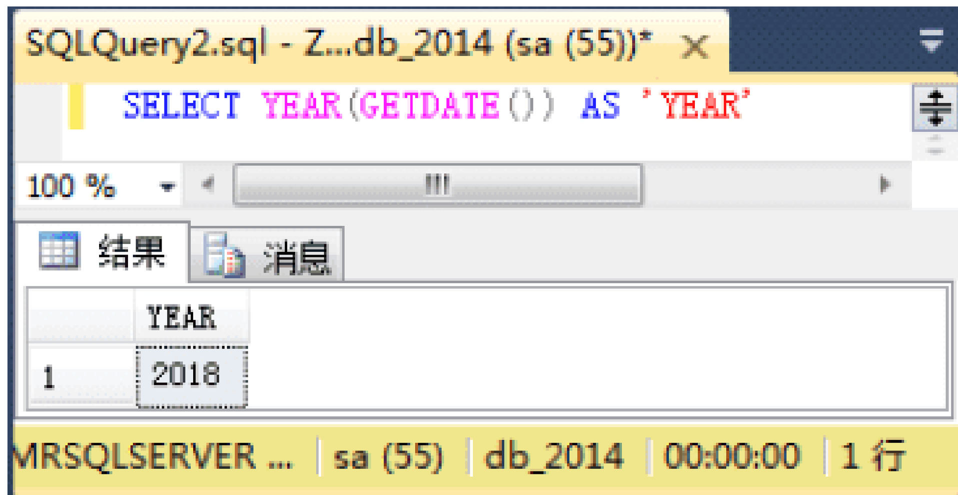


图 6.36 返回当前日期的“年份”

SQL 语句如下：

```
SELECT YEAR(GETDATE()) AS 'YEAR'
```

6.4.6 DATEDIFF（返回日期和时间的边界数）函数

DATEDIFF 函数用于返回日期和时间的边界数。
语法格式如下：

```
DATEDIFF(datepart,startdate,enddate)
```

参数说明如下。

☑ datepart：规定了应在日期的哪一部分计算差额的参数。
☑ startdate：表示计算的开始日期，startdate 是返回 datetime 值、smalldatetime 值或日期格式字符串的表达式。
☑ enddate：表示计算的终止日期。enddate 是返回 datetime 值、smalldatetime 值或日期格式字符串的表达式。

SQL Server 识别的日期部分和缩写如表 6.9 所示。

表 6.9 日期部分和缩写对照表

日 期 部 分	缩　　写	日 期 部 分	缩　　写
year	yy,yyyy	week	wk, ww
quarter	qq, q	hour	hh
month	mm, m	minute	mi, n
dayofyear	dy, y	second	ss, s
day	dd, d	millisecond	ms

有关 DATEDIFF 函数使用的几点说明如下。

☑ startdate 是从 enddate 中减去。如果 startdate 比 enddate 晚，则返回负值。
☑ 当结果超出整数值范围，DATEDIFF 产生错误。对于毫秒，最大数是 24 天 20 小时 31 分钟零 23.647 秒。对于秒，最大数是 68 年。
☑ 计算跨分钟、秒和毫秒这些边界的方法，使得 DATEDIFF 给出的结果在全部数据类型中是一致的。结果是带正负号的整数值，其等于跨第一个和第二个日期间的 datepart 边界数。例如，

在 1 月 4 日（星期日）和 1 月 11 日（星期日）之间的星期数是 1。

【例 6.38】　使用 DATEDIFF 函数，返回两个日期之间的天数，SQL 语句及运行结果如图 6.37 所示。（**实例位置：资源包\源码\06\6.38**）

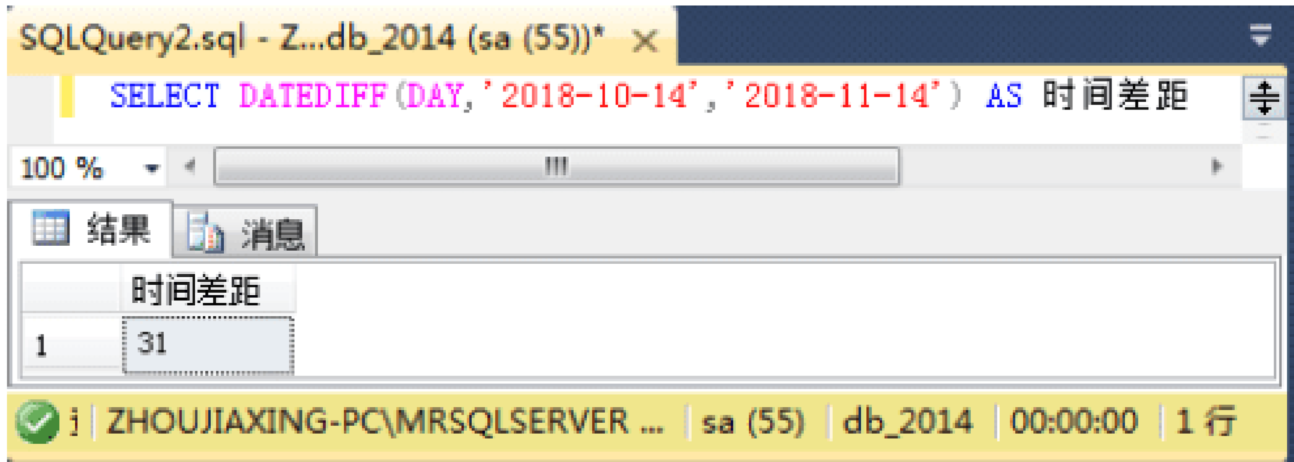


图 6.37　返回两个日期之间的天数

SQL 语句如下：

```
SELECT DATEDIFF(DAY,'2018-10-14','2018-11-14') AS 时间差距
```

6.4.7　DATEADD（添加日期时间）函数

DATEADD 函数将表示日期或时间间隔的数值与日期中指定的日期部分相加后，返回一个新的 DT_DBTIMESTAMP 值。number 参数的值必须为整数，而 date 参数的取值必须为有效日期。

语法格式如下：

```
DATEADD(datepart, number, date)
```

参数说明如下。

☑　datepart：指定要与数值相加的日期部分的参数。

☑　number：用于与 datepart 相加的值。该值必须是分析表达式时已知的整数值。

☑　date：返回有效日期或日期格式的字符串的表达式。

SQL Server 识别的日期部分和缩写如表 6.9 所示。

注意

如果指定一个不是整数的值，则将废弃此值的小数部分。

【例 6.39】　使用 DATEADD 函数，在现在时间上加上一个月，SQL 语句及运行结果如图 6.38 所示。（**实例位置：资源包\源码\06\6.39**）

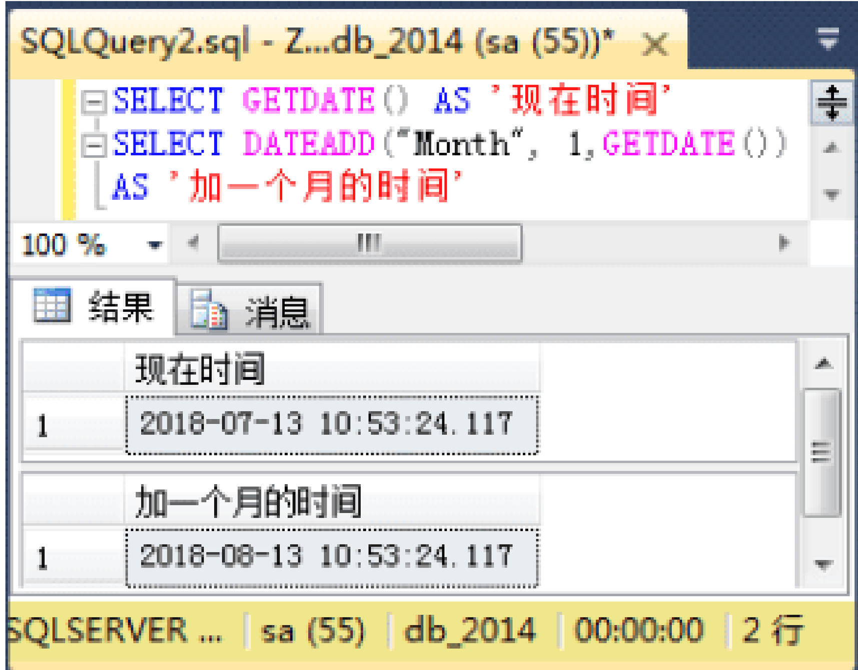


图 6.38　将现在时间上加上一个月

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
SELECT DATEADD("Month", 1,GETDATE())
AS '加一个月的时间'
```

【例 6.40】 使用 DATEADD 函数，在现在时间上加上两天，SQL 语句及运行结果如图 6.39 所示。（**实例位置：资源包\源码\06\6.40**）

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
SELECT DATEADD("DAY", 2,GETDATE())
AS '加两天的时间'
```

【例 6.41】 使用 DATEADD 函数，在现在时间上加上一年，SQL 语句及运行结果如图 6.40 所示。（**实例位置：资源包\源码\06\6.41**）

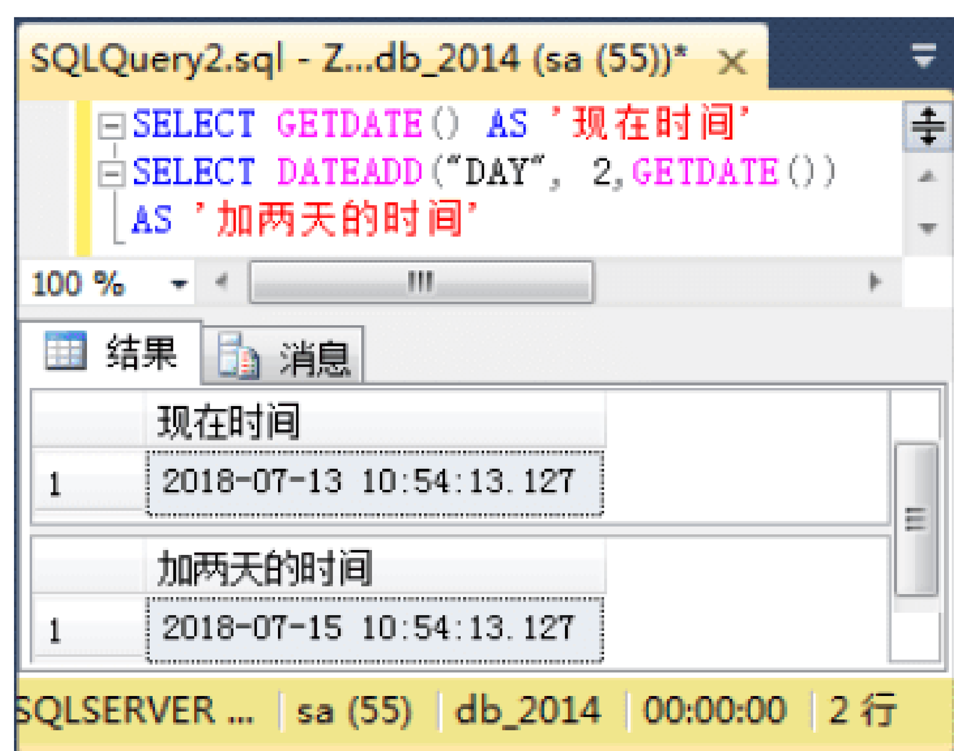


图 6.39　将现在时间上加两天

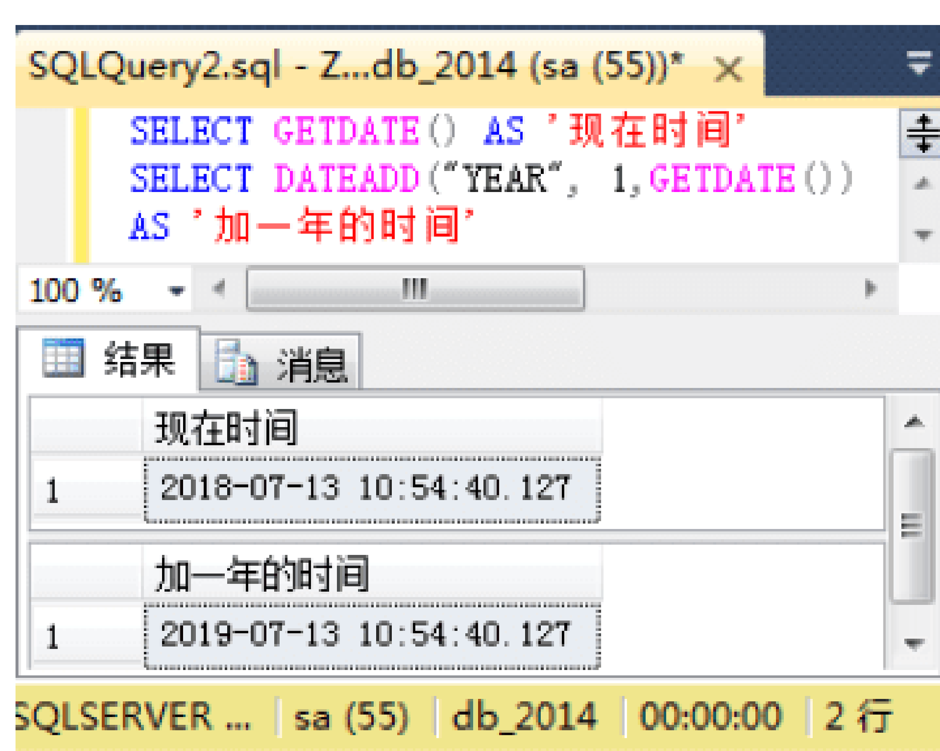


图 6.40　将现在时间上加上一年

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
SELECT DATEADD("YEAR", 1,GETDATE())
AS '加一年的时间'
```



6.5　转 换 函 数

如果 SQL Server 没有自动执行数据类型的转换，可以使用 CAST 和 CONVERT 转换函数将一种数据类型的表达式转换为另一种数据类型的表达式。例如，如果比较 char 和 datetime 表达式、smallint 和 int 表达式或不同长度的 char 表达式，则 SQL Server 自动对这些表达式进行转换。

6.5.1　转换函数概述

当遇到类型转换的问题时，可以使用 SQL Server 提供的 CAST 和 CONVERT 函数。这两种函数不

但可以将指定的数据类型转换为另一种数据类型，还可用来获得各种特殊的数据格式。CAST 和 CONVERT 函数都可用于选择列表、WHERE 子句和允许使用表达式的任何地方。

在 SQL Server 中数据类型转换分为两种，分别如下。

☑ 隐性转换：SQL Server 自动处理某些数据类型的转换。例如，如果比较 char 和 datetime 表达式、smallint 和 int 表达式或不同长度的 char 表达式，SQL Server 可将它们自动转换，这种转换称为隐性转换，对这些转换不必使用 CAST 函数。

☑ 显示转换：显示转换是指 CAST 和 CONVERT 函数，CAST 和 CONVERT 函数将数值从一种数据类型（局部变量、列或其他表达式）转换到另一种数据类型。



说明

隐性转换对用户是不可见的，SQL Server 自动将数据从一种数据类型转换成另一种数据类型。例如，如果一个 smallint 变量和一个 int 变量相比较，这个 smallint 变量在比较前即被隐性转换成 int 变量。

有关转换函数使用的几点说明如下。

☑ CAST 函数基于 SQL-92 标准并且优先于 CONVERT。

☑ 当从一个 SQL Server 对象的数据类型向另一个数据类型转换时，一些隐性和显式数据类型转换是不支持的。例如，nchar 数值根本就不能被转换成 image 数值。nchar 只能显式地转换成 binary，隐性地转换到 binary 是不支持的。nchar 可以显式地或者隐性地转换成 nvarchar。

☑ 当处理 sql_variant 数据类型时，SQL Server 支持将具有其他数据类型的对象隐性转换成 sql_variant 类型。然而，SQL Server 并不支持从 sql_variant 数据类型隐性地转换到其他数据类型的对象。

6.5.2　CAST 函数

CAST 函数用于将某种数据类型的表达式显式转换为另一种数据类型。

语法格式如下：

```
CAST(expression AS data_type)
```

参数说明如下。

☑ expression：表示任何有效的 SQL Server 表达式

☑ AS：用于分隔两个参数，在 AS 之前的是要处理的数据，在 AS 之后是要转换的数据类型。

☑ data_type：表示目标系统所提供的数据类型，包括 bigint 和 sql_variant，不能使用用户定义的数据类型。

使用 CAST 函数进行数据类型转换时，在下列情况下能够被接受。

☑ 两个表达式的数据类型完全相同。

☑ 两个表达式可隐性转换。

☑ 必须显式转换数据类型。

如果试图进行不可能的转换（例如，将含有字母的 char 表达式转换为 int 类型），SQL Server 将显示一条错误信息。

如果转换时没有指定数据类型的长度，则 SQL Server 自动提供长度为 30。

【例 6.42】 使用 CAST 函数将字符串 MINGRIKEJI 转换为 NVARCHAR(6)类型，SQL 语句及运行结果如图 6.41 所示。（**实例位置：资源包\源码\06\6.42**）

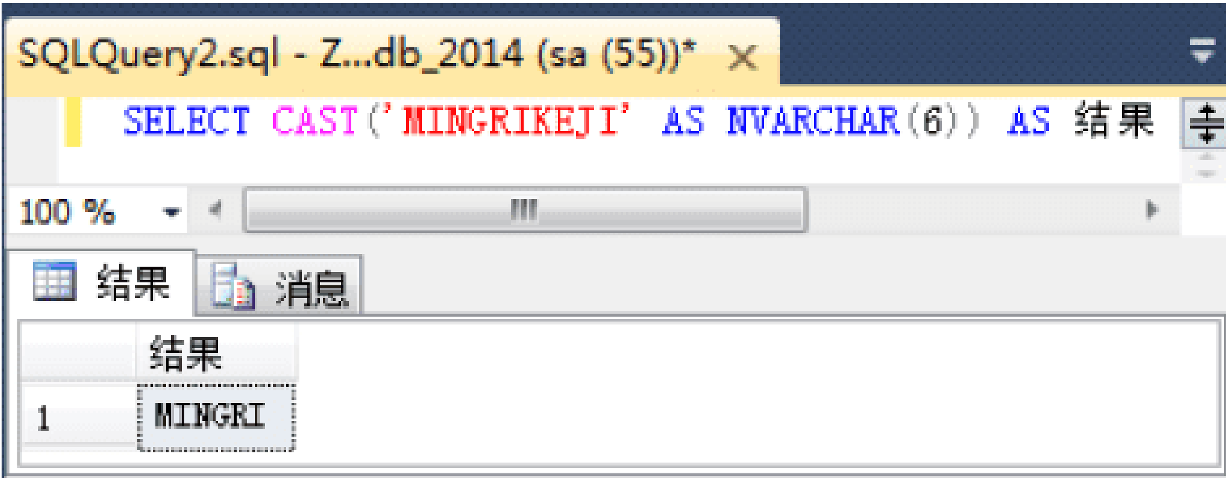


图 6.41 使用 CAST 函数转换字符串

SQL 语句如下：

```
SELECT CAST('MINGRIKEJI' AS NVARCHAR(6)) AS 结果
```

6.5.3 CONVERT 函数

CONVERT 函数与 CAST 函数的功能相似。该函数不是一个 ANSI 标准 SQL 函数，它可以按照指定的格式将数据转换为另一种数据类型。

语法格式如下：

```
CONVERT(data_type[(length)],expression [, style])
```

参数说明如下。

☑ data_type：表示目标系统所提供的数据类型，包括 bigint 和 sql_variant。不能使用用户定义的数据类型。

☑ length：为 nchar、nvarchar、char、varchar、binary 和 varbinary 数据类型的可选参数。参数 expression 表示任何有效的 SQL Server 表达式。

☑ style：为日期样式，指定当将 datetime 数据转换为某种字符数据时或将某种字符数据转换为 datetime 数据时会使用 style 中的样式。

style 日期样式如表 6.10 所示。

表 6.10 style 日期样式

样　式	说　明	输入/输出格式
0 或 100（*）	默认值	mon dd yyyy hh:mi AM（或者 PM）
1/101	美国	mm/dd/yyyy
2/102	ANSI	yy.mm.dd
3/103	英国/法国	dd/mm/yy
4/104	德国	dd.mm.yy

续表

样　　式	说　　明	输入/输出格式
5/105	意大利	dd-mm-yy
6/106	-	dd mon yy
7/107	-	mon dd,yy
8/108	-	hh:mm:ss
9 或 109（*）	默认值+毫秒	mon dd yyyy hh:mi:ss:mmmAM（或者 PM）
10 或 110	美国	mm-dd-yy
11 或 111	日本	yy/mm/dd
12 或 112	ISO	yymmdd
13 或 113（*）	欧洲默认值+毫秒	dd mon yyyy hh:mm:ss:mmm（24h）
14 或 114	-	hh:mi:ss:mmm（24h）
20 或 120（*）	ODBC 规范	yyyy-mm-dd hh:mm:ss（24h）
21 或 121（*）	ODBC 规范（带毫秒）	yyyy-mm-dd hh:mm:ss.mmm（24h）
126	ISO 8601	yyyy-mm-dd Thh:mm:ss:mmm（不含空格）
130	科威特	dd mon yyyy hh:mi:ss:mmmAM（或者 PM）
131	科威特	dd/mm/yy hh:mi:ss.mmmAM（或者 PM）

【例 6.43】　显示当前日期和时间，并使用 CAST 函数将当前日期和时间改为字符数据类型，然后使用 CONVERT 函数以 ISO 8601 格式显示日期和时间，SQL 语句及运行结果如图 6.42 所示。（**实例位置：资源包\源码\06\6.43**）

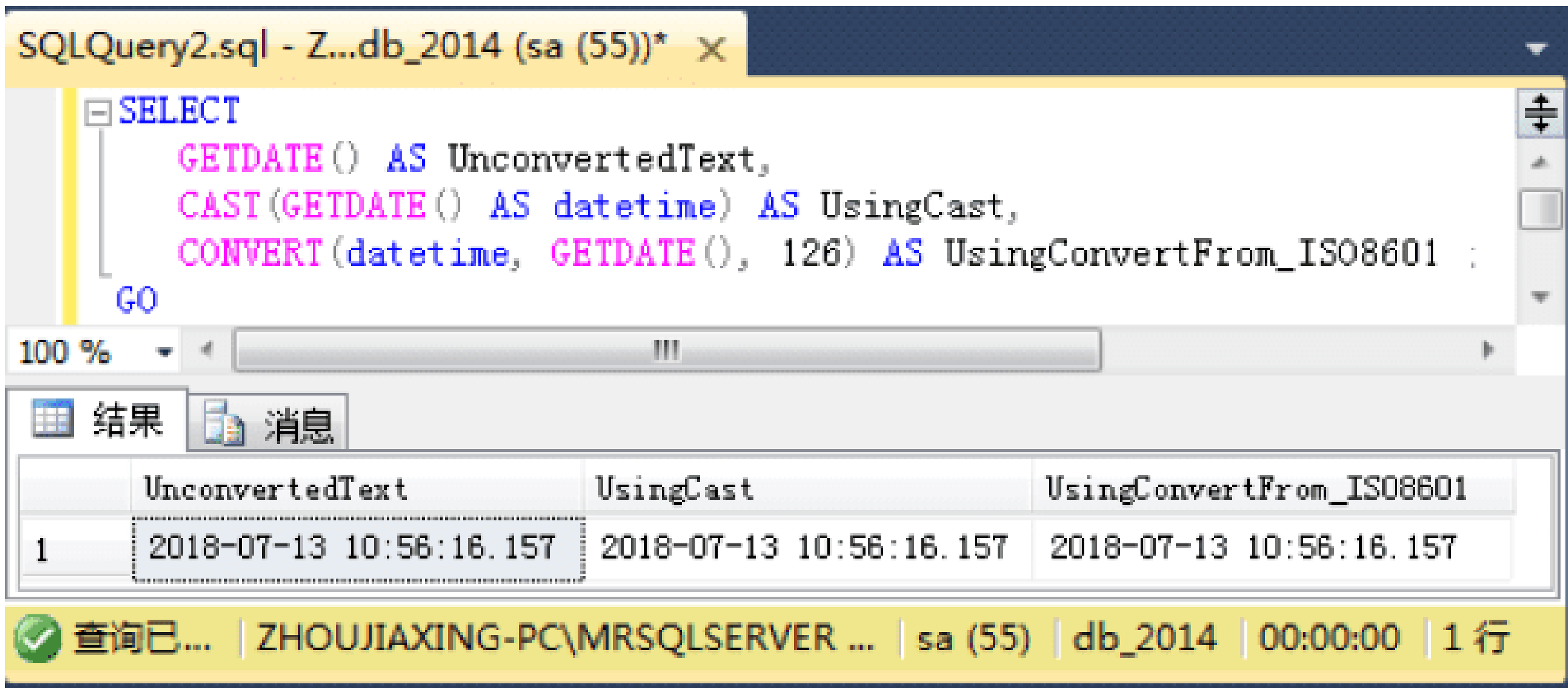


图 6.42　转换数据类型（1）

SQL 语句如下：

```
SELECT
    GETDATE() AS UnconvertedText,
    CAST(GETDATE() AS datetime) AS UsingCast,
    CONVERT(datetime, GETDATE(), 126) AS UsingConvertFrom_ISO8601;
GO
```

【例 6.44】　将当前日期和时间显示为字符数据，并使用 CAST 函数将字符数据改为 datetime 数据类型，然后使用 CONVERT 函数将字符数据改为 datetime 数据类型，SQL 语句及运行结果如图 6.43

所示。（**实例位置：资源包\源码\06\6.44**）

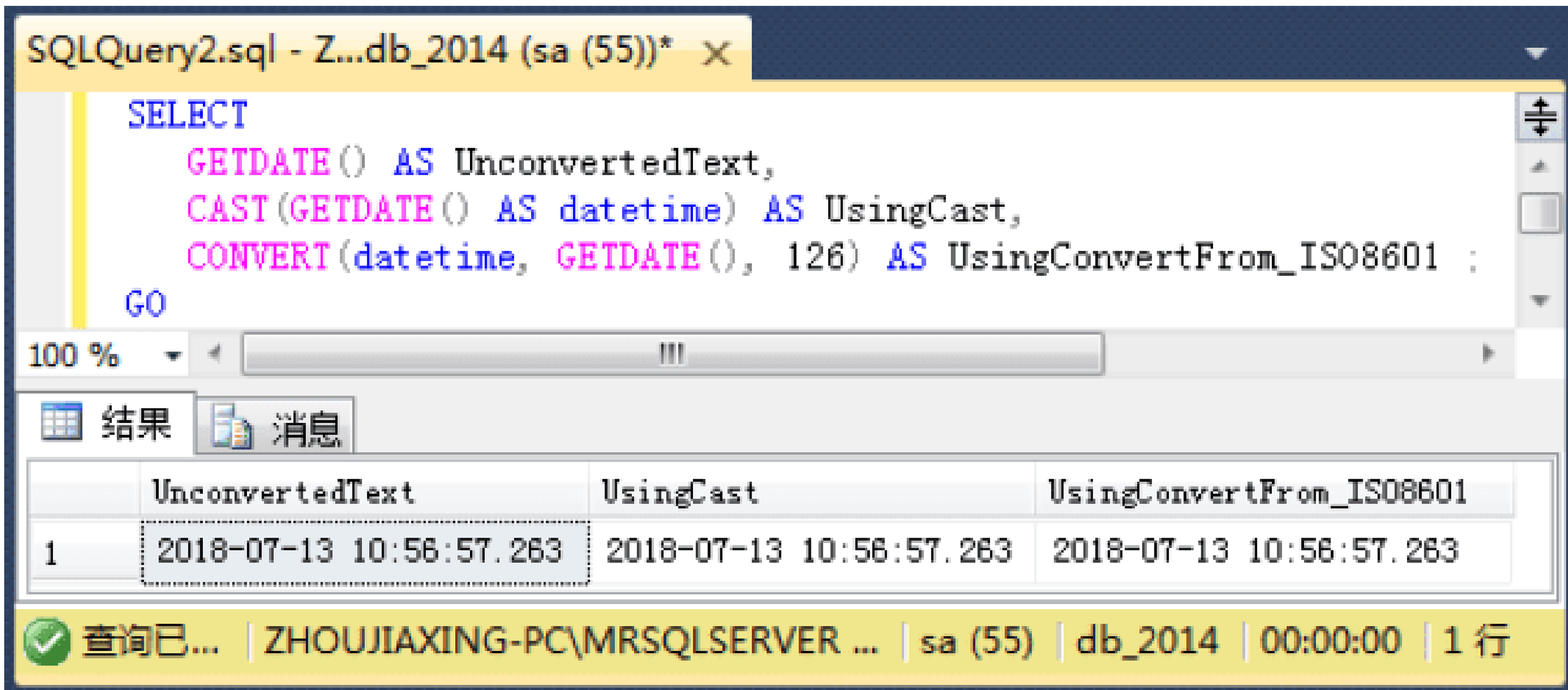


图 6.43　转换数据类型（2）

SQL 语句如下：

```
SELECT
    GETDATE() AS UnconvertedText,
    CAST(GETDATE() AS datetime) AS UsingCast,
    CONVERT(datetime, GETDATE(), 126) AS UsingConvertFrom_ISO8601;
GO
```



6.6　元数据函数

元数据函数主要是返回与数据库相关的信息，本节向读者介绍 COL_LENGTH、COL_NAME 和 DB_NAME 3 个常用的元数据函数。

6.6.1　元数据函数概述

元数据函数描述了数据的结构和意义，它主要用于返回数据库中的相应信息，其中包括以下方面。

☑　返回数据库中数据表或视图的个数和名称。

☑　返回数据表中数据字段的名称、数据类型、长度等描述信息。

☑　返回数据表中定义的约束、索引、主键或外键等信息。

常用的元数据函数及说明如表 6.11 所示。

表 6.11　常用的元数据函数及说明

函 数 名 称	说　　明
COL_LENGTH	返回列的定义长度（以字节为单位）
COL_NAME	返回数据库列的名称，该列具有相应的表标识号和列标识号
DB_NAME	返回数据库名
OBJECT_ID	返回数据库对象标识号

6.6.2　COL_LENGTH 函数

COL_LENGTH 函数用于返回列的定义长度。

语法格式如下：

```
COL_LENGTH('table', 'column')
```

参数 table 表示数据表名称，参数 column 表示数据表的列名称。

【例 6.45】　首先创建一个数据表，然后使用 COL_LENGTH 函数返回指定列定义的长度，SQL 语句及运行结果如图 6.44 所示。（**实例位置：资源包\源码\06\6.45**）

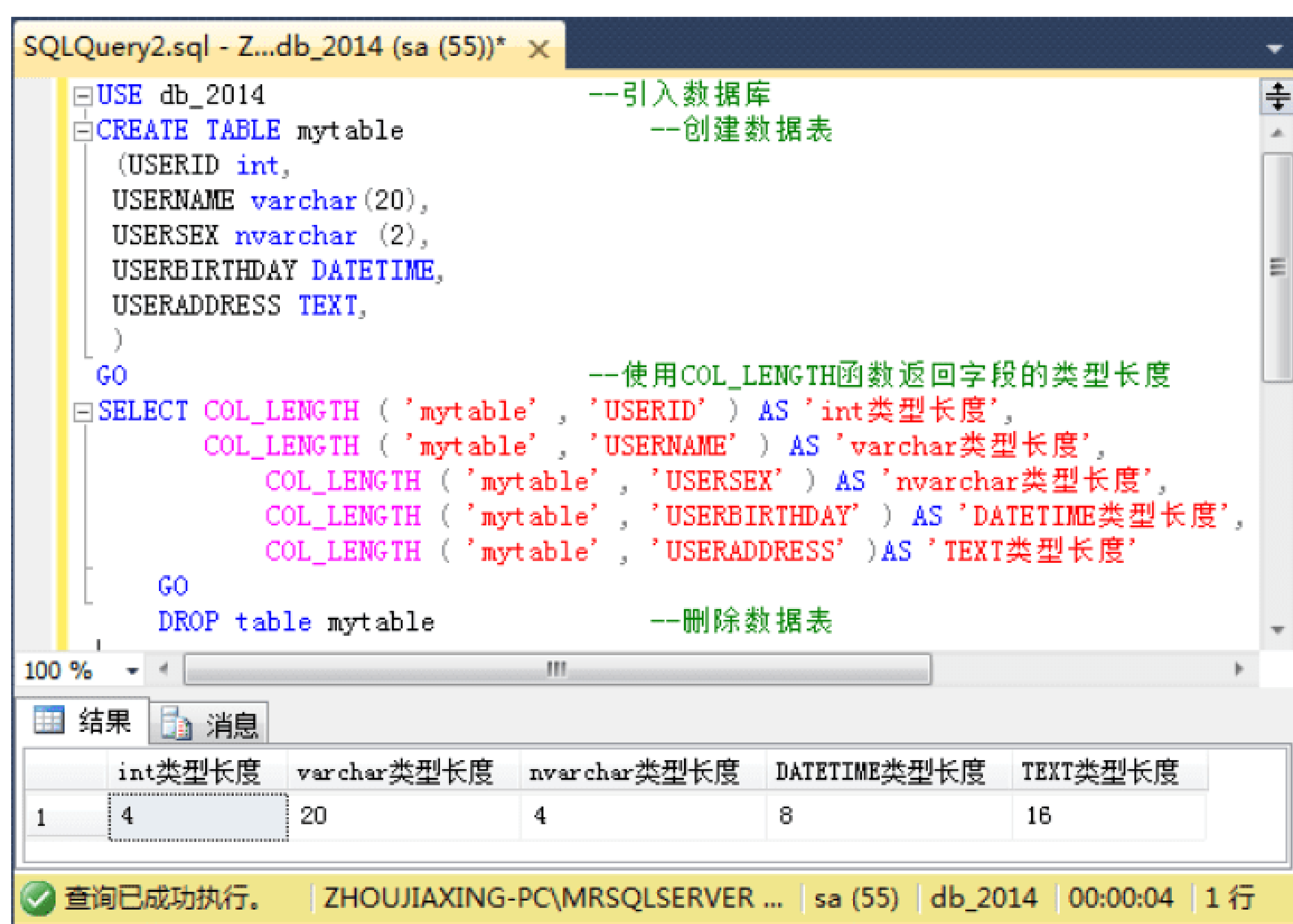


图 6.44　返回字段类型的长度

SQL 语句如下：

```
USE db_2014                                     --引入数据库
CREATE TABLE mytable                            --创建数据表
 (USERID int,
 USERNAME varchar(20),
 USERSEX nvarchar(2),
 USERBIRTHDAY DATETIME,
 USERADDRESS TEXT,
 )
GO                                              --使用 COL_LENGTH 函数返回字段的类型长度
SELECT COL_LENGTH('mytable', 'USERID') AS 'int 类型长度',
       COL_LENGTH('mytable', 'USERNAME') AS 'varchar 类型长度',
       COL_LENGTH('mytable', 'USERSEX') AS 'nvarchar 类型长度',
       COL_LENGTH('mytable', 'USERBIRTHDAY') AS 'DATETIME 类型长度',
       COL_LENGTH('mytable', 'USERADDRESS') AS 'TEXT 类型长度'
```

```
    GO
    DROP table mytable                                    --删除数据表
```

6.6.3 COL_NAME 函数

COL_NAME 函数用于返回数据库列的名称。

语法格式如下：

```
COL_NAME(table_id, column_id)
```

参数说明如下。

☑ table_id：包含数据库列的表的标识号，table_id 属于 int 类型。

☑ column_id：表示列的标识号，column_id 属于 int 类型。

【例 6.46】 使用 COL_NAME 函数，返回 db_2014 数据库的 Employee 表中首列的名称，SQL 语句及运行结果如图 6.45 所示。（**实例位置：资源包\源码\06\6.46**）

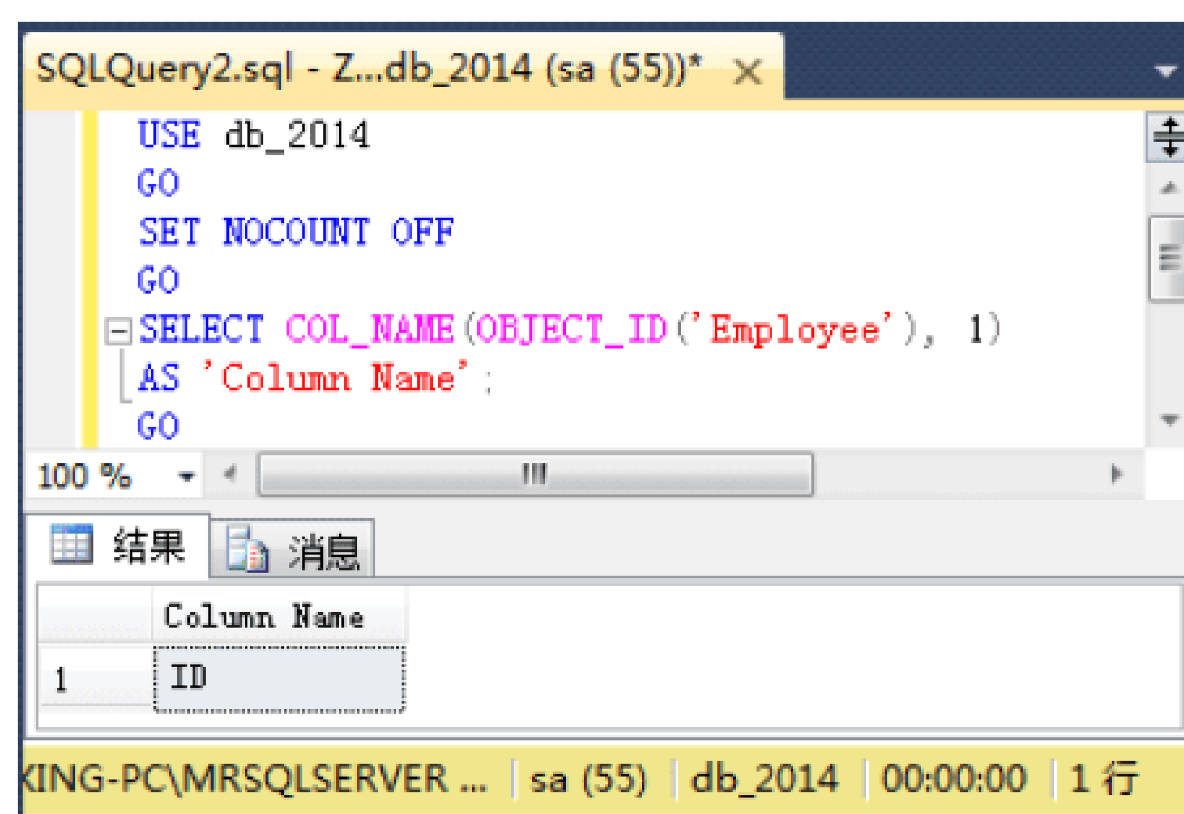


图 6.45 返回 Employee 表中首列的名称

SQL 语句如下：

```
USE db_2014
GO
SET NOCOUNT OFF
GO
SELECT COL_NAME(OBJECT_ID('Employee'), 1)
AS 'Column Name';
GO
```

6.6.4 DB_NAME 函数

DB_NAME 函数返回数据库名称。

语法格式如下：

```
DB_NAME([database_id])
```

参数说明如下。

☑ database_id：要返回的数据库的标识号（ID）。database_id 的数据类型为 int，无默认值。如果未指定 ID，则返回当前数据库名称。

☑ 返回类型：nvarchar(128)类型。

【例 6.47】 使用 DB_NAME 函数，返回当前数据库的名称，SQL 语句及运行结果如图 6.46 所示。（**实例位置：资源包\源码\06\6.47**）

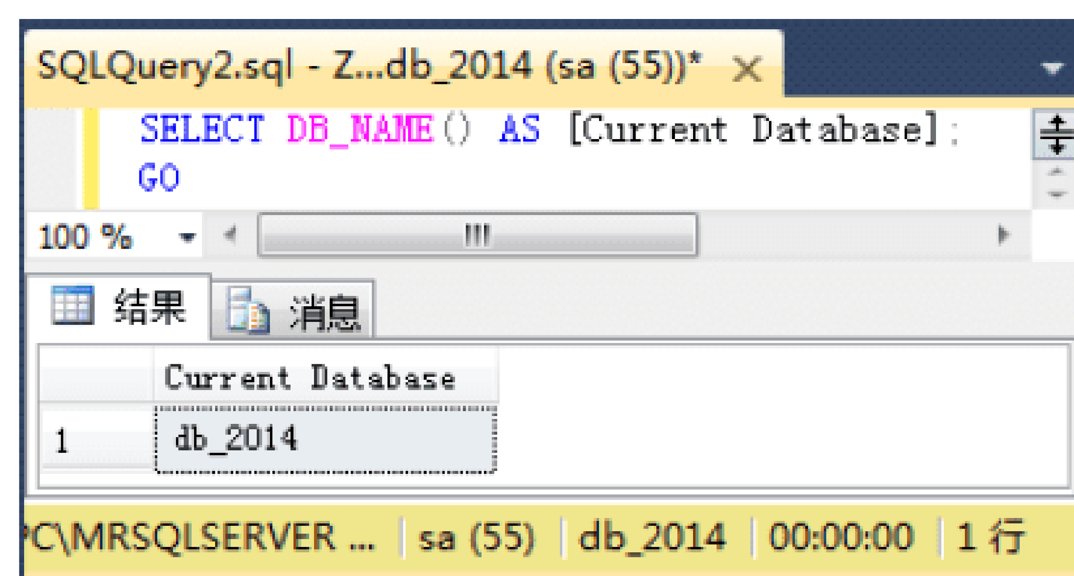


图 6.46　返回当前数据库的名称

SQL 语句如下：

```
SELECT DB_NAME() AS [Current Database];
GO
```

6.7　小　　结

本章主要对 SQL 中常用的函数进行了讲解，并通过具体的实例说明了各个函数的使用方法。通过本章的学习，读者应该能够掌握常用的 SQL 函数及其使用方法，并能够在实际应用中使用这些 SQL 函数提高工作的效率。

第 7 章

视图操作

（视频讲解：15 分钟）

本章主要介绍视图的操作，包括视图概述，以及从视图中浏览数据、向视图中添加数据、修改视图中的数据、删除视图中的数据等视图中的数据操作相关知识。通过本章的学习，读者将掌握创建或者删除视图的使用方法，能够使用视图来优化数据。

学习摘要：

- 视图的概念
- 界面方式创建视图
- 命令方式创建视图
- 视图中的数据操作



7.1　视 图 概 述

视图中的内容是由查询定义来的，并且视图和查询都是通过 SQL 语句定义的，它们有着许多相同和不同之处。具体如下。

☑ 存储：视图存储为数据库设计的一部分，而查询则不是。视图可以禁止所有用户访问数据库中的基表，而要求用户只能通过视图操作数据。这种方法可以保护用户和应用程序不受某些数据库修改的影响，同样也可以保护数据表的安全性。

☑ 排序：可以排序任何查询结果，但是只有当视图包括 TOP 子句时才能排序视图。

☑ 加密：可以加密视图，但不能加密查询。

7.1.1　界面方式操作视图

1．视图的创建

下面在 SQL Server Management Studio 中创建视图 View_Stu，具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2014。

（3）鼠标右键单击“视图”选项，在弹出的快捷菜单中选择“新建视图”命令，如图 7.1 所示。

（4）进入“添加表”窗体，如图 7.2 所示。在列表框中选择学生信息表 Student，单击“添加”按钮，然后单击“关闭”按钮关闭该窗体。

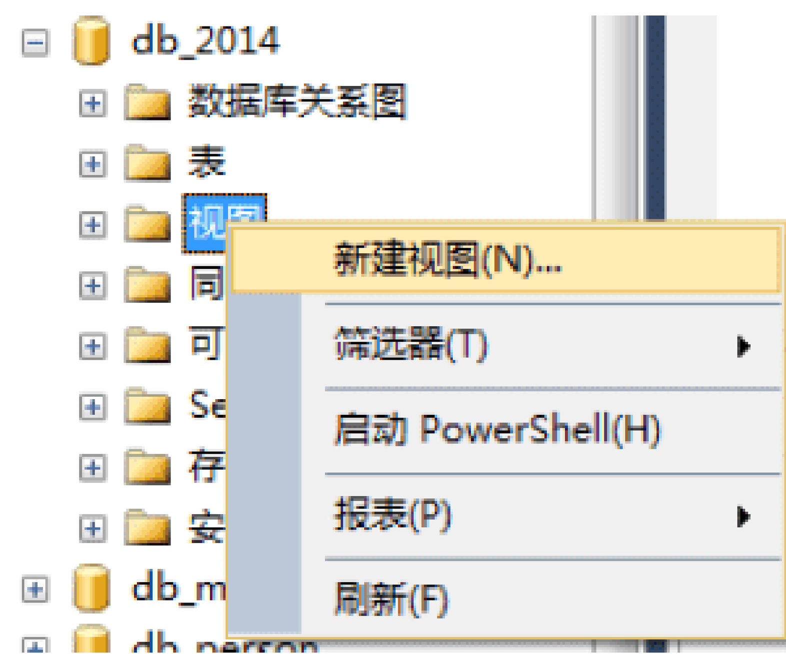


图 7.1　新建视图

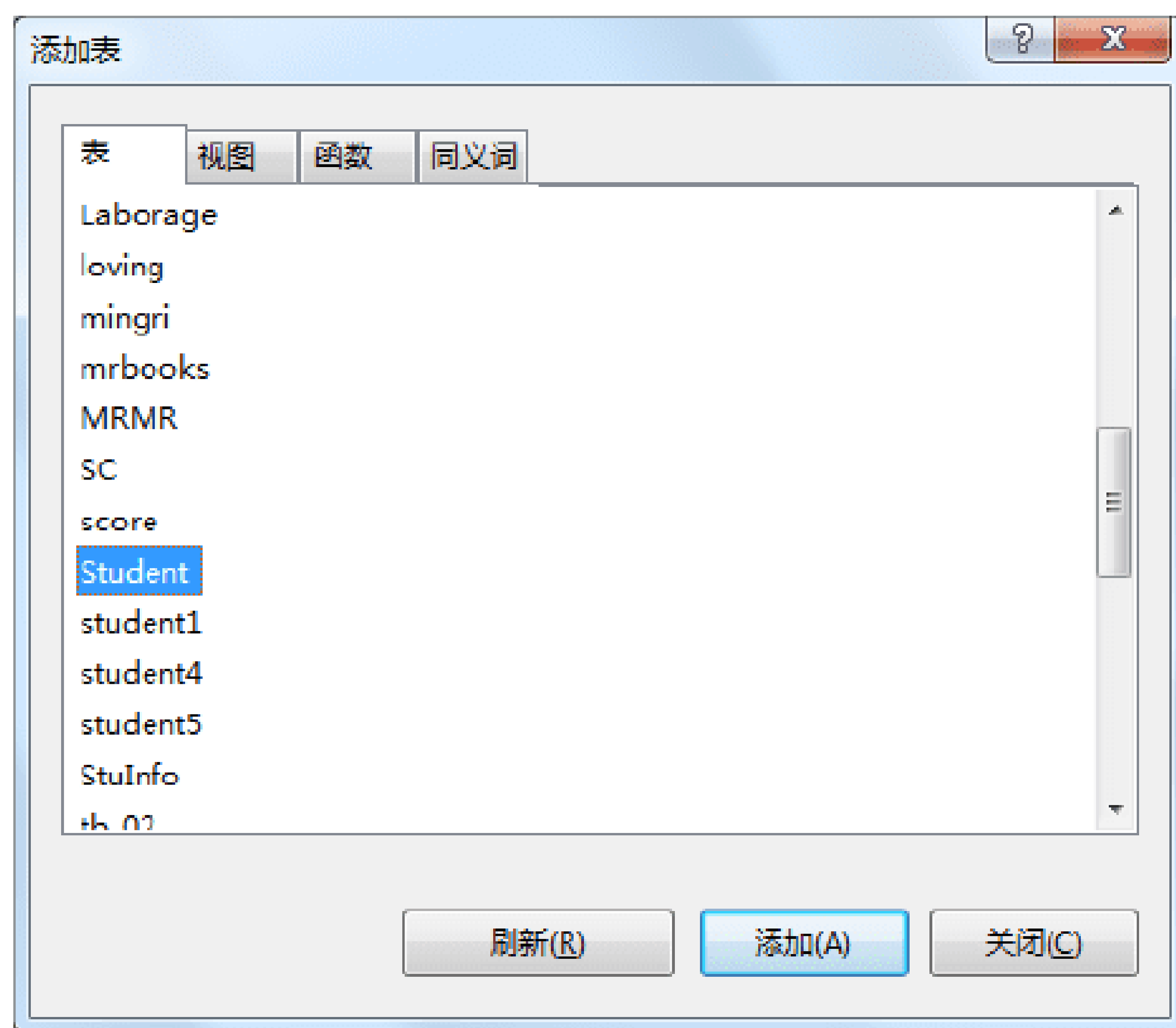


图 7.2　“添加表”窗体

（5）进入视图设计窗口，如图 7.3 所示。在“表选择区”中选择“所有列”选项，单击执行按钮，视图结果区中自动显示视图结果。

（6）单击工具栏中的“保存”按钮，弹出“选择名称”对话框，如图 7.4 所示。在“输入视图名称”文本框中输入视图名称 View_student，单击“确定”按钮即可保存该视图。

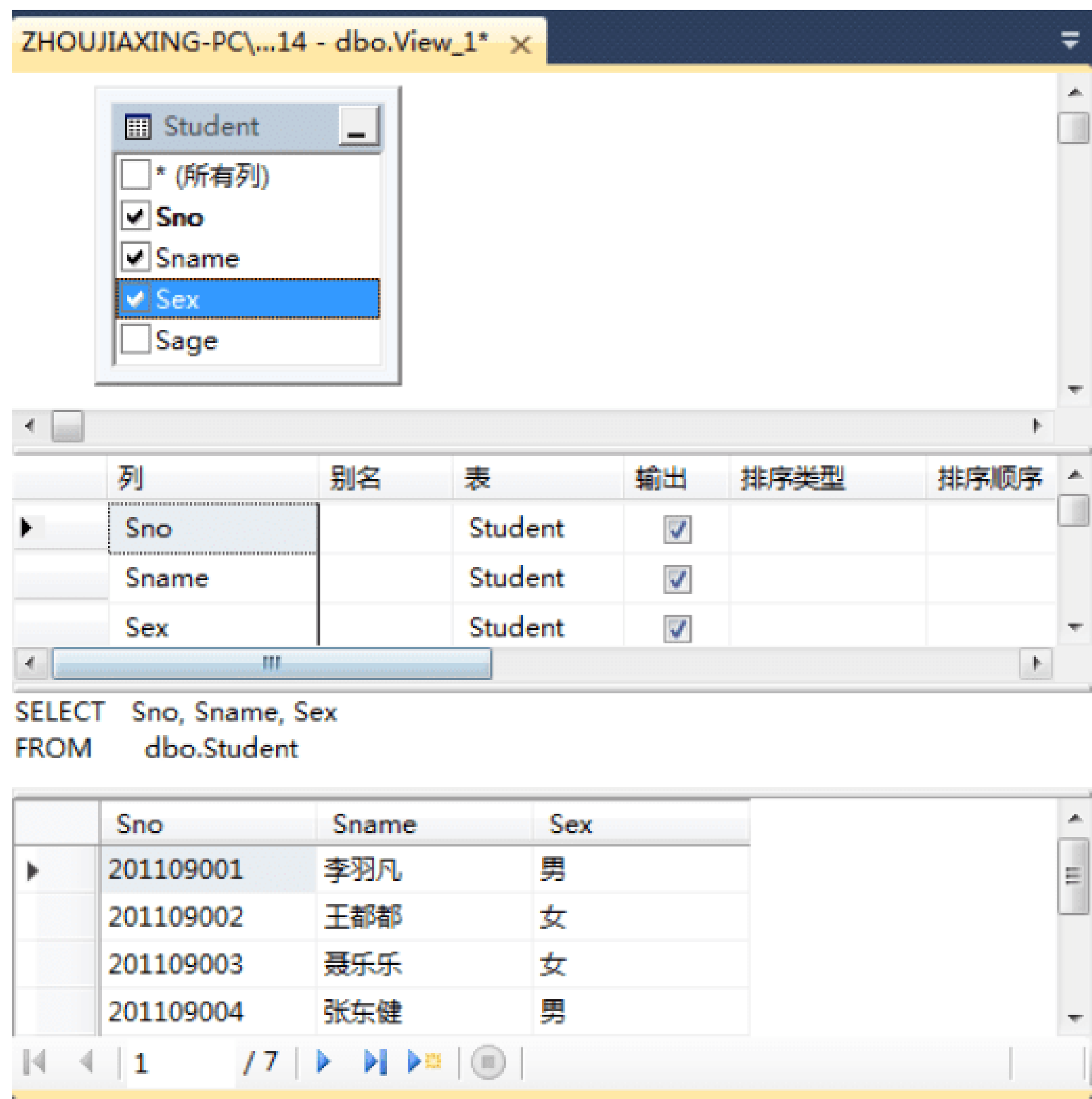


图 7.3　视图设计窗口

2．视图的删除

用户可以删除视图。删除视图时，底层数据表不受影响，但会造成与该视图关联的权限丢失。

下面介绍如何在 SQL Server Management Studio 中删除视图，具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2014。

（3）展开“视图”节点，鼠标右键单击要删除的视图 View_student，在弹出的快捷菜单中选择“删除”命令，如图 7.5 所示。

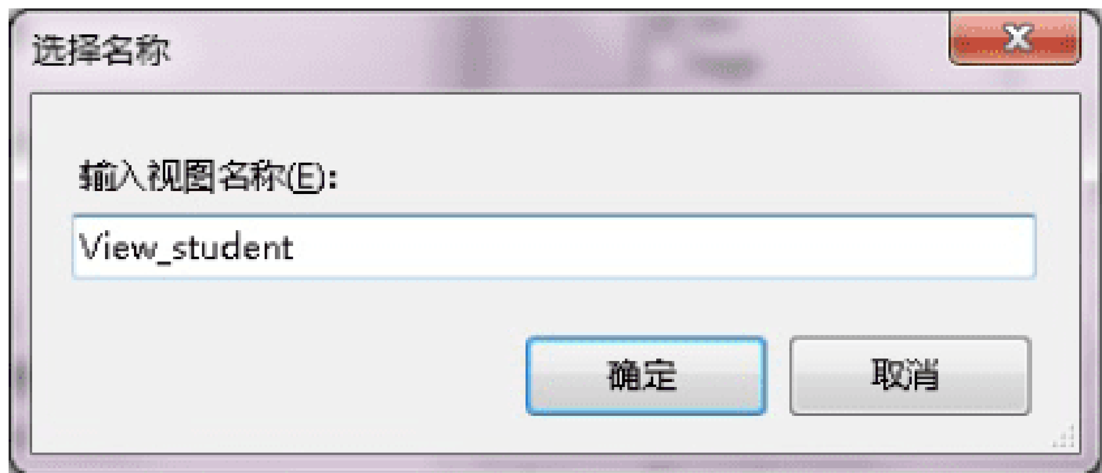


图 7.4　“选择名称”对话框

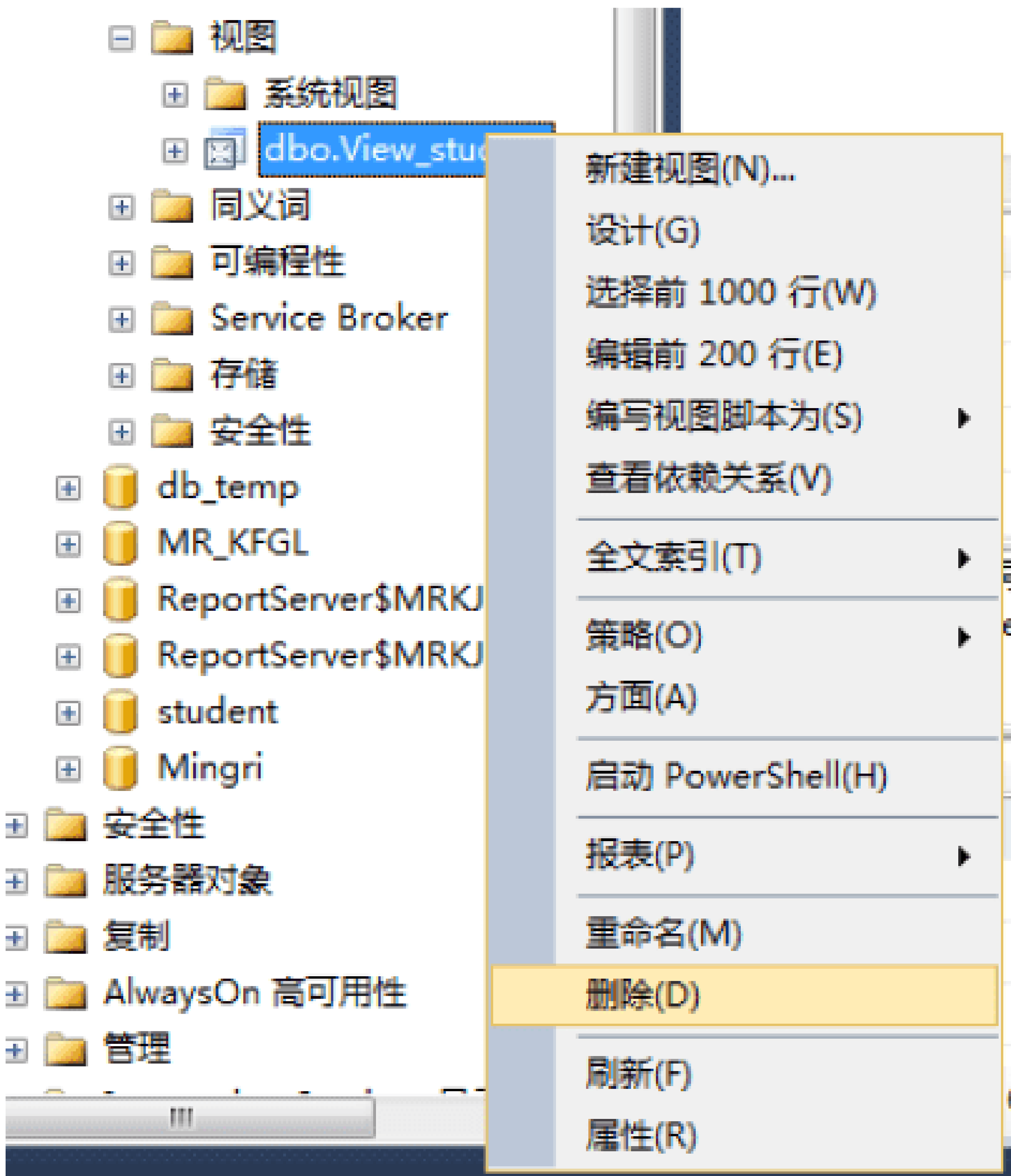


图 7.5　删除视图

（4）在弹出的“删除对象”对话框中单击“确定”按钮，即可删除该视图。

7.1.2　使用 CREATE VIEW 语句创建视图

使用 CREATE VIEW 语句可以创建视图，语法格式如下：

```
CREATE VIEW [schema_name .] view_name [(column [,...n])]
[WITH <view_attribute> [,...n]]
AS select_statement [;]
[WITH CHECK OPTION]
<view_attribute> ::=
{
  [ENCRYPTION] [SCHEMABINDING] [VIEW_METADATA]
}
```

参数如表 7.1 所示。

表 7.1　CREATE VIEW 语句参数说明

参　　数	说　　明
schema_name	视图所属架构的名称
view_name	视图的名称。视图名称必须符合有关标识符的规则。可以选择是否指定视图所有者名称
column	视图中的列使用的名称
AS	指定视图要执行的操作
select_statement	定义视图的 SELECT 语句
CHECK OPTION	强制针对视图执行的所有数据修改语句都必须符合在 select_statement 中设置的条件
ENCRYPTION	对视图进行加密
SCHEMABINDING	将视图绑定到基础表的架构
VIEW_METADATA	指定为引用视图的查询请求浏览模式的元数据时，SQL Server 实例将向 DB-Library、ODBC 和 OLE DB API 返回有关视图的元数据信息，而不返回基表的元数据信息

【例 7.01】　创建仓库入库表视图。（**实例位置：光盘\源码\07\7.01**）

代码如下：

```
CREATE VIEW view_1
AS
SELECT * FROM tb_joinDepot
```

7.1.3　使用 ALTER VIEW 语句修改视图

使用 ALTER VIEW 语句可以修改视图，语法格式如下：

```
ALTER VIEW view_name [(column [,...n])]
[WITH ENCRYPTION]
AS
```

```
select_statement
[WITH CHECK OPTION]
```

参数说明如下。

☑ view_name：要更改的视图。
☑ column：一列或多列的名称，用逗号分开，将成为给定视图的一部分。
☑ n：表示 column 可重复 n 次的占位符。
☑ WITH ENCRYPTION：加密 syscomments 表中包含 ALTER VIEW 语句文本的条目。使用 WITH ENCRYPTION 可防止将视图作为 SQL Server 复制的一部分发布。
☑ AS：视图要执行的操作。
☑ select_statement：定义视图的 SELECT 语句。
☑ WITH CHECK OPTION：强制视图上执行的所有数据的修改语句都必须符合由定义视图的 select_statement 设置的准则。


说明

如果原来的视图定义是用 WITH ENCRYPTION 或 CHECK OPTION 创建的，那么只有在 ALTER VIEW 中也包含这些选项时，这些选项才有效。

【例 7.02】 修改仓库入库表视图。（**实例位置：光盘\源码\07\7.02**）

关键代码如下：

```
ALTER VIEW View_1(oid,wareName)
AS
SELECT oid,wareName
FROM tb_joinDepot
WHERE id=9
--查看视图定义
EXEC sp_helptext 'View_1'
```

7.1.4 使用 DROP VIEW 语句删除视图

使用 DROP VIEW 语句可以删除视图，语法格式如下：

```
DROP VIEW view_name [,...n]
```

参数说明如下。

☑ view_name：要删除的视图名称。视图名称必须符合标识符规则。可以选择是否指定视图所有者名称。若要查看当前创建的视图列表，使用 sp_help。
☑ n：表示可以指定多个视图的占位符。


注意

在单击“全部除去”按钮删除视图以前，可以在“除去对象”对话框中单击“显示相关性”按钮，即可查看该视图依附的对象，以确认该视图是否为想要删除的视图。

【例 7.03】　使用 Transact-SQL 删除视图。（**实例位置：光盘\源码\07\7.03**）

（1）首先单击“新建查询”按钮。

（2）在查询编辑器窗口中输入以下代码，单击工具栏上的执行按钮。此时执行查询结果将在下面的子窗口中显示出来。相关代码如下：

```
USE db_2014
GO
DROP VIEW View_1
GO
```



7.2　视图中的数据操作

7.2.1　从视图中浏览数据

下面在 SQL Server Management Studio 中查看视图 View_Stu 的信息，具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2014。

（3）再依次展开“视图”节点，就会显示出当前数据库中的所有视图，鼠标右键单击要查看信息的视图。

（4）如果想要查看视图的属性，在弹出的快捷菜单中选择“属性”命令，如图 7.6 所示，弹出“视图属性”窗口，如图 7.7 所示。

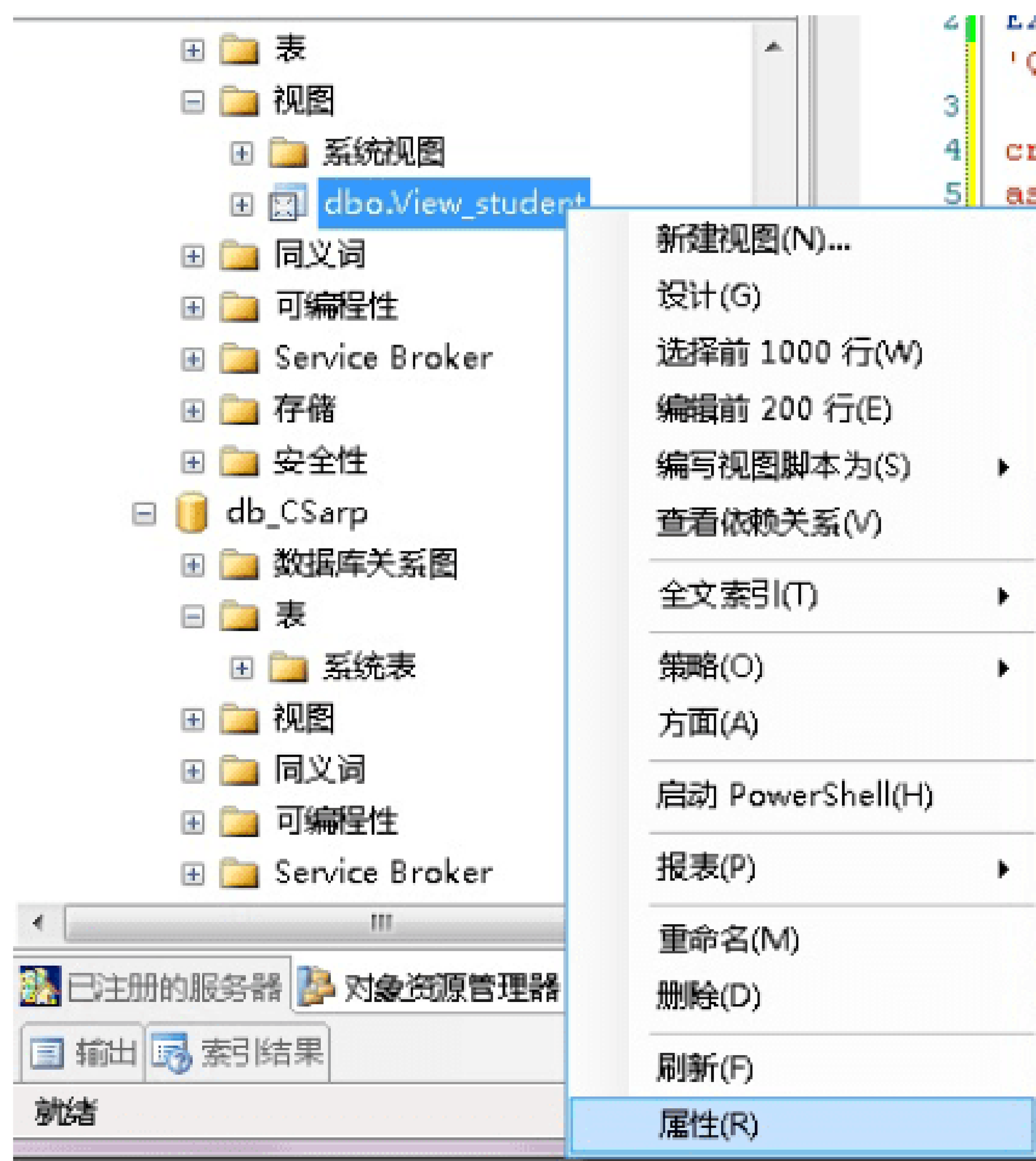


图 7.6　查看视图属性

（5）如果想要查看视图中的内容，可在如图 7.6 所示的快捷菜单中选择“编辑前 200 行”命令，在右侧即可以显示视图中的内容，如图 7.8 所示。

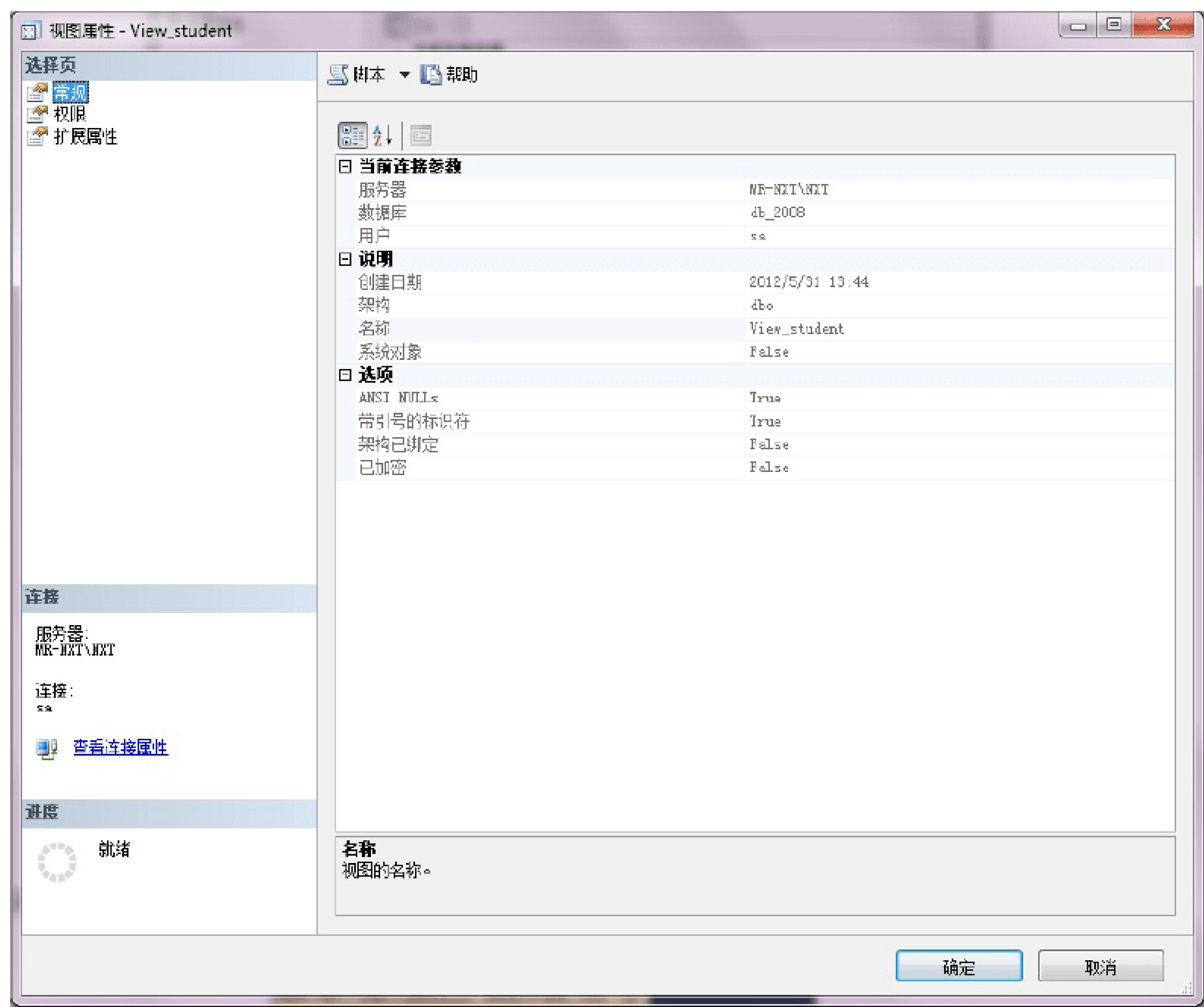


图 7.7 “视图属性”窗口

（6）如果想要重新设置视图，可在如图 7.6 所示的快捷菜单中选择“设计”命令，弹出视图设计窗口，如图 7.9 所示。在此窗口中可对视图进行重新设置。


MR-NXT\NXT.db_20... dbo.View_student

学号	姓名	性别
201109003	聂乐乐	女
201109004	张东健	男
201109004	张东健	男
201109001	李羽凡	男
201109001	李羽凡	男
201109002	王嘟嘟	女
201109005	王子	男

图 7.8 显示视图中的内容

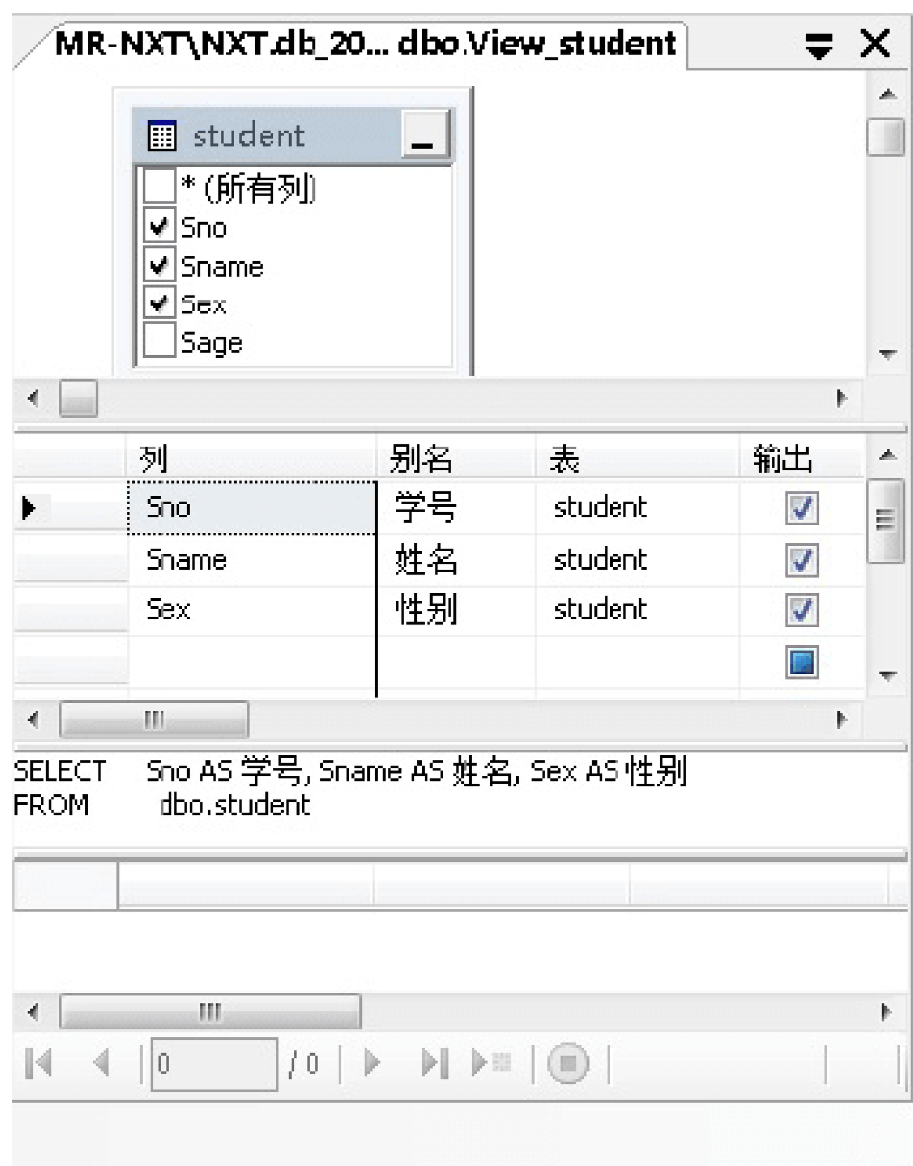


图 7.9 视图设计窗口

7.2.2 向视图中添加数据

使用视图可以添加新的记录，但应该注意的是，新添加的数据实际上是存储在与视图相关的表中。

例如，向视图 View_student 中插入信息“20110901，明日科技，女”。步骤如下。

（1）鼠标右键单击要插入记录的视图，在弹出的快捷菜单中选择“设计”命令，显示视图设计窗口。

（2）在显示视图结果的最下面一行直接输入新记录即可，如图 7.10 所示。

（3）然后按 Enter 键，即可把信息插入视图中。

（4）单击 ! 按钮，完成新记录的添加，如图 7.11 所示。

学号	姓名	性别
22050120	刘春芬	女
22050121	刘丽	女
22050125	刘小宁	男
20110901	明日科技	女

图 7.10　插入记录

学号	姓名	性别
20047109	鸿飞	男
20049110	秀丽	女
20110901	明日科技	女
22050110	张晓亮	男
22050111	李壮	

插入的记录

图 7.11　插入记录后的视图

7.2.3　修改视图中的数据

使用视图可以修改数据记录，但是与插入记录相同，修改的是数据表中的数据记录。

例如，修改视图 View_student 中的记录，将“明日科技”修改为“明日”。步骤如下。

（1）鼠标右键单击要修改记录的视图，在弹出的快捷菜单中选择“设计”命令，显示视图设计窗口。

（2）在显示的视图结果中，选择要修改的内容，直接修改即可。

（3）最后按 Enter 键，即可把信息保存到视图中。

7.2.4　删除视图中的数据

使用视图可以删除数据记录，但是与插入记录相同，删除的是数据表中的数据记录。

例如，删除视图 View_student 中的记录“明日科技”。步骤如下。

（1）鼠标右键单击要删除记录的视图，在弹出的快捷菜单中选择“设计”命令，显示视图设计窗口。

（2）在显示视图的结果中，鼠标右键单击要删除的行“明日科技”，在弹出的快捷菜单中选择“删除”命令，弹出删除视图中的数据对话框，如图 7.12 所示。

（3）单击“是”按钮，便将该记录删除。



图 7.12　删除视图中的数据对话框

7.3　小　　结

本章介绍了创建视图、修改视图和删除视图的方法。读者可以针对表创建视图并能够通过视图实现对表的操作以及查看视图是否存在，修改视图中的内容等。

第 8 章

Transact-SQL 语法基础

（视频讲解：29 分钟）

本章主要介绍 Transact-SQL（T-SQL）语法基础。Transact-SQL 是标准 SQL 程序设计语言的增强版，是应用程序与 SQL Server 数据库引擎沟通的主要语言。不管应用程序的用户接口是什么，都会通过 Transact-SQL 语句与 SQL Server 数据库引擎进行沟通。

学习摘要：

- **Transact-SQL 概述**
- **常量、变量**
- **注释符、运算符和通配符**



8.1　Transact-SQL 概述

8.1.1　Transact-SQL 语言的组成

Transact-SQL 语言是具有强大查询功能的数据库语言，除此以外，Transact-SQL 还可以控制 DBMS 为其用户提供的所有功能，主要包括如下。

- ☑ 数据定义语言（Data Definition Language，DDL）：SQL 让用户定义存储数据的结构和组织，以及数据项之间的关系。
- ☑ 数据检索语言：SQL 允许用户或应用程序从数据库中检索存储的数据并使用它。
- ☑ 数据操作语言（Data Manipulation Language，DML）：SQL 允许用户或应用程序通过添加新数据、删除旧数据和修改以前存储的数据对数据库进行更新。
- ☑ 数据控制语言（Data Control Language，DCL）：可以使用 SQL 来限制用户检索、添加和修改数据的能力，保护存储的数据不被未授权的用户所访问。
- ☑ 数据共享：可以使用 SQL 来协调多个并发用户共享数据，确保它们不会相互干扰。
- ☑ 数据完整性：SQL 在数据库中定义完整性约束条件，使它不会由不一致的更新或系统失败而遭到破坏。

因此，Transact-SQL 是一种综合性语言，用来控制并与数据库管理系统进行交互作用。Transact-SQL 是数据库子语言，包含大约 40 条专用于数据库管理任务的语句。各类的 SQL 语句分别如表 8.1～表 8.5 所示。

数据操作类 SQL 语句如表 8.1 所示。

表 8.1　数据操作类 SQL 语句

语　　句	功　　能
SELECT	从数据库表中检索数据行和列
INSERT	把新的数据记录添加到数据库中
DELETE	从数据库中删除数据记录
UPDATE	修改现有的数据库中的数据

数据定义类 SQL 语句如表 8.2 所示。

表 8.2　数据定义类 SQL 语句

语　　句	功　　能
CREATE TABLE	在一个数据库中创建一个数据库表
DROP TABLE	从数据库删除一个表
ALTER TABLE	修改一个现存表的结构
CREATE VIEW	把一个新的视图添加到数据库中
DROP VIEW	从数据库中删除视图

续表

语　句	功　能
CREATE INDEX	为数据库表中的一个字段构建索引
DROP INDEX	从数据库表中的一个字段中删除索引
CREATE PROCEDURE	在一个数据库中创建一个存储过程
DROP PROCEDURE	从数据库中删除存储过程
CREATE TRIGGER	创建一个触发器
DROP TRIGGER	从数据库中删除触发器
CREATE SCHEMA	向数据库添加一个新模式
DROP SCHEMA	从数据库中删除一个模式
CREATE DOMAIN	创建一个数据值域
ALTER DOMAIN	改变域定义
DROP DOMAIN	从数据库中删除一个域

数据控制类 SQL 语句如表 8.3 所示。

表 8.3　数据控制类 SQL 语句

语　句	功　能
GRANT	授予用户访问权限
DENY	拒绝用户访问
REVOKE	删除用户访问权限

事务控制类 SQL 语句如表 8.4 所示。

表 8.4　事务控制类 SQL 语句

语　句	功　能
COMMIT	结束当前事务
ROLLBACK	中止当前事务
SET TRANSACTION	定义当前事务数据访问特征

程序化 SQL 语句如表 8.5 所示。

表 8.5　程序化 SQL 语句

语　句	功　能
DECLARE	定义查询游标
EXPLAN	描述查询描述数据访问计划
OPEN	检索查询结果打开一个游标
FETCH	检索一条查询结果记录
CLOSE	关闭游标
PREPARE	为动态执行准备 SQL 语句
EXECUTE	动态地执行 SQL 语句
DESCRIBE	描述准备好的查询

8.1.2　Transact-SQL 语句结构

每条 SQL 语句均由一个谓词（Verb）开始，该谓词描述这条语句要产生的动作，如 SELECT 或 UPDATE 关键字。谓词后紧接着一个或多个子句（Clause），子句中给出了被谓词作用的数据或提供谓词动作的详细信息。每一条子句都由一个关键字开始。下面以 SELECT 语句为例介绍 Transact-SQL 语句的结构，语法格式如下：

```
SELECT　子句
[INTO 子句]
FROM 子句
[WHERE 子句]
[GROUP　BY 子句]
[HAVING　子句]
[ORDER BY 子句]
```

【例 8.01】　在 student 数据库中查询 course 表的信息。在查询编辑器窗口中运行的结果如图 8.1 所示。（**实例位置：资源包\源码\06\8.01**）

a.sql - MR-NXT\NXT.student (sa (52))*

```
use student
select * from course where 课程类别='艺术类' order by 课程内容
```

结果　消息

	课程代号	课程名称	课程类别	课程内容
1	k01	喜爱的逻辑	艺术类	童年
2	k02	喜爱的逻辑	艺术类	童年2
3	k01	个人单曲	艺术类	LOVING
4	k04	经典金曲	艺术类	0.5公里

图 8.1　查询 course 数据表的信息

SQL 语句如下：

```
use student
select * from course where 课程类别='艺术类' order by 课程内容
```

8.2　常　　量


视频讲解

常量也叫常数，常量是指在程序运行过程中不发生改变的量。它可以是任何数据类型，本节将对常量使用进行详细讲解。

1．字符串常量

字符串常量定义在单引号内。字符串常量包含字母、数字字符（a～z、A～Z 和 0～9）及特殊字符

（如数字号#、感叹号！、at 符@）。

例如，以下为字符串常量：

```
'Hello World'
'Microsoft Windows'
'Good Morning'
```

2．二进制常量

在 Transact-SQL 中定义二进制常量，需要使用 0x，并采用十六进制来表示，不再需要括号。

例如，以下为二进制常量：

```
0xB0A1
0xB0C4
0xB0C5
```

3．bit 常量

在 Transact-SQL 中，bit 常量使用数字 0 或 1 即可，并且不包括在引号中。如果使用一个大于 1 的数字，则该数字将转换为 1。

4．日期和时间常量

定义日期和时间常量需要使用特定格式的字符日期值，并使用单引号。

例如，以下为日期和时间常量：

```
'2008 年 1 月 9 日'
'15:39:15'
'01/09/2008'
'06:59   AM'
```



视频讲解

8.3 变　　量

数据在内存中存储可以变化的量叫作变量。为了在内存存储信息，用户必须指定存储信息的单元，并为该存储单元命名，以方便获取信息，这就是变量的功能。Transact-SQL 可以使用两种变量：一种是局部变量；另外一种是全局变量。局部变量和全局变量的主要区别在于存储的数据作用范围不一样，本节将对变量的使用进行详细讲解。

8.3.1 局部变量

局部变量是用户可自定义的变量，它的作用范围仅在程序内部。局部变量的名称是用户自定义的，命名的局部变量名要符合 SQL Server 标识符命名规则，局部变量名必须以@开头。

1．声明局部变量

局部变量的声明需要使用 DECLARE 语句。语法格式如下：

```
DECLARE
{
@varaible_name datatype [,… n]
}
```

参数说明如下。

☑ @varaible_name：局部变量的变量名必须以@开头，另外变量名的形式必须符合 SQL Server 标识符的命名方式。

☑ datatype：局部变量使用的数据类型可以是除 text、ntext 或者 image 类型外所有的系统数据类型和用户自定义数据类型。一般来说，如果没有特殊的用途，建议在应用时尽量使用系统提供的数据类型。这样做可以减少维护应用程序的工作量。

例如，声明局部变量@songname，SQL 语句如下：

```
DECLARE @songname char(10)
```

2．为局部变量赋值

为变量赋值的方式一般有两种：一种是使用 SELECT 语句；一种是使用 SET 语句。使用 SELECT 语句为变量赋值的语法格式如下：

```
SELECT @varible_name = expression
[FROM table_name [,...n]
WHERE clause]
```

上面的 SELECT 语句的作用是为了给变量赋值，而不是为了从表中查询出数据。而且在使用 SELECT 语句进行赋值的过程中，并不一定非要使用 FROM 关键字和 WHERE 子句。

【例 8.02】 在 student 数据库的 course 表中，把“课程内容”是“艺术类”信息赋值给局部变量@songname，并把它的值用 print 关键字显示出来。在查询编辑器窗口中运行的结果如图 8.2 所示。（**实例位置：资源包\源码\06\8.02**）

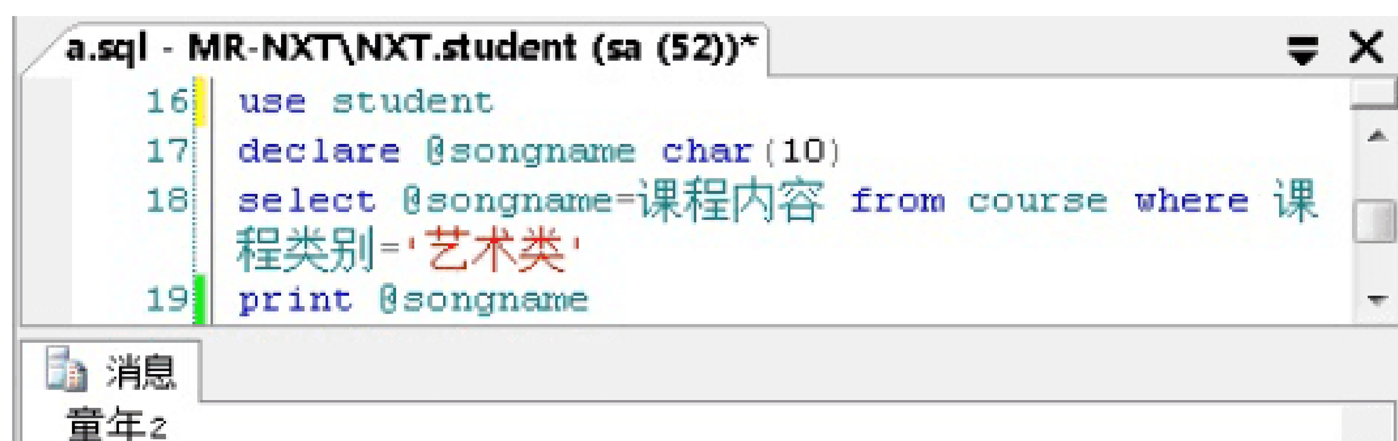


图 8.2　把查询内容赋值给局部变量

SQL 语句如下：

```
use student
declare @songname char(10)
select @songname=课程内容 from course where 课程类别='艺术类'
print @songname
```

SELECT 语句赋值和查询不能混淆，例如，声明一个局部变量名是@b 并赋值的 SQL 语句如下：

```
DECLARE @b int
SELECT @b=1
```

另一种为局部变量赋值的方式是使用 SET 语句。使用 SET 语句对变量进行赋值的常用语法格式如下：

```
{ SET @varible_name = ecpression } [,... n]
```

下面是一个简单的赋值语句：

```
DECLARE @song char(20)
SET @song = 'I love flower'
```

还可以为多个变量一起赋值，相应的 SQL 语句如下：

```
DECLARE @b int, @c char(10),@a int
SELECT @b=1, @c='love',@a=2
```



注意

数据库语言和编程语言有一些关键字，关键字是在某一环境下能够促使某一操作发生的字符组。为避免冲突和产生错误，在命名表、列、变量以及其他对象时应避免使用关键字。

8.3.2　全局变量

全局变量是 SQL Server 系统内部事先定义好的变量，不需要用户参与定义，对用户而言，其作用范围并不局限于某一程序，而是任何程序均可随时调用。全局变量通常用于存储一些 SQL Server 的配置设定值和效能统计数据。

SQL Server 一共提供了 30 多个全局变量，本节只对一些常用变量的功能和使用方法进行介绍。全局变量的名称都是以@@开头的。

（1）@@CONNECTIONS

记录自最后一次服务器启动以来，所有针对这台服务器进行的连接数目，包括没有连接成功的尝试。使用@@CONNECTIONS 可以让系统管理员很容易地得到今天所有试图连接本服务器的连接数目。

（2）@@CUP_BUSY

记录自上次启动以来尝试的连接数，无论连接成功还是失败，都以 ms 为单位的 CPU 工作时间。

（3）@@CURSOR_ROWS

返回在本次服务器连接中，打开游标取出数据行的数目。

（4）@@DBTS

返回当前数据库中 timestamp 数据类型的当前值。

（5）@@ERROR

返回执行上一条 Transact-SQL 语句所返回的错误代码。

在 SQL Server 服务器执行完一条语句后，如果该语句执行成功，将返回@@ERROR 的值为 0，如果该语句执行过程中发生错误，将返回错误的信息，而@@ERROR 将返回相应的错误编号，该编号将一直保持下去，直到下一条语句得到执行为止。

由于@@ERROR 在每一条语句执行后被清除并且重置，应在语句验证后立即检查它，或将其保存到一个局部变量中以备事后查看。

【例 8.03】　在 pubs 数据库中修改 authors 数据表时，用@@ERROR 检测限制查询冲突。在查询编辑器窗口中运行的结果如图 8.3 所示。（**实例位置：资源包\源码\06\8.03**）



图 8.3　修改数据时检测错误

SQL 语句如下：

```
use pubs
GO
UPDATE authors SET au_id = '172 32 1176'
WHERE au_id = '172-32-1176'
IF @@ERROR = 547
PRINT 'A check constraint violation occurred'
```

（6）@@FETCH_STATUS

返回上一次使用游标 FETCH 操作所返回的状态值，且返回值为整型。

返回值描述如表 8.6 所示。

表 8.6　@@FETCH_STATUS 返回值的描述

返　回　值	描　　述
0	FETCH 语句成功
−1	FETCH 语句失败或此行不在结果集中
−2	被提取的行不存在

例如，到了最后一行数据后，还要接着取下一行数据，返回的值为−2，表示返回的值已经丢失。

（7）@@IDENTITY

返回最近一次插入的 identity 列的数值，返回值是 numeric。

【例 8.04】　在 pugs 数据库的 jobs 数据表中，插入一行数据，并用@@identity 显示新行的标识值。在查询编辑器窗口运行的结果如图 8.4 所示。（**实例位置：资源包\源码\06\8.04**）

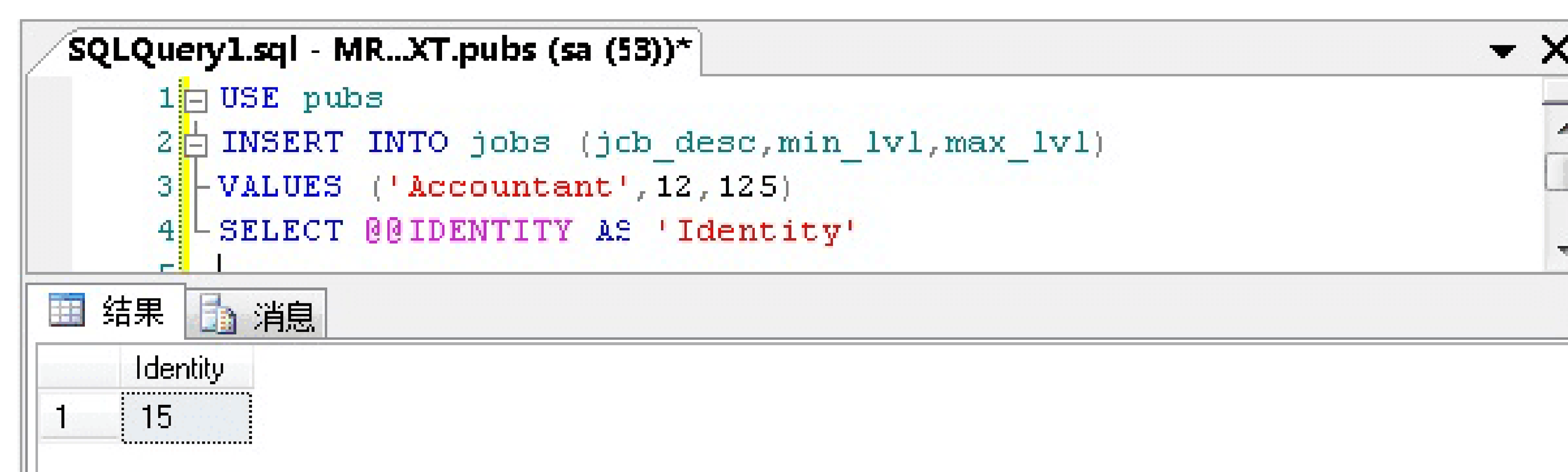


图 8.4　显示新行的标识值

SQL 语句如下：

```
USE pubs
INSERT INTO jobs (job_desc,min_lvl,max_lvl)
VALUES ('Accountant',12,125)
SELECT @@IDENTITY AS 'Identity'
```

（8）@@IDLE

返回以 ms 为单位计算 SQL Server 服务器自最近一次启动以来处于停顿状态的时间。

（9）@@IO_BUSY

返回以 ms 为单位计算的 SQL Server 服务器自最近一次启动以来花在输入和输出上的时间。

（10）@@LOCK_TIMEOUT

返回当前对数据锁定的超时设置。

（11）@@PACK_RECEIVED

返回 SQL Server 服务器自最近一次启动以来一共从网络上接收数据分组的数目。

（12）@@PACK_SENT

返回 SQL Server 服务器自最近一次启动以来一共向网络上发送数据分组的数目。

（13）@@PROCID

返回当前存储过程的 ID 标识。

（14）@@REMSERVER

返回在登录记录中记载远程 SQL Server 服务器的名字。

（15）@@ROWCOUNT

返回上一条 SQL 语句所影响到数据行的数目。对所有不影响数据库数据的 SQL 语句，这个全局变量返回的结果是 0。在进行数据库编程时，经常要检测@@ROWCOUNT 的返回值，以便明确所执行的操作是否达到了目标。

（16）@@SPID

返回当前服务器进程的 ID 标识。

（17）@@TOTAL_ERRORS

返回自 SQL Server 服务器启动来，所遇到读写错误的总数。

（18）@@TOTAL_READ

返回自 SQL Server 服务器启动来，读磁盘的次数。

（19）@@TOTAL_WRITE

返回自 SQL Server 服务器启动来，写磁盘的次数。

（20）@@TRANCOUNT

返回当前连接中，处于活动状态事务的数目。

（21）@@VERSION

返回当前 SQL Server 服务器安装日期、版本，以及处理器的类型。



8.4　注释符、运算符与通配符

8.4.1　注释符（Annotation）

注释语句不是可执行语句，不参与程序的编译，通常是一些说明性的文字，对代码的功能或者代码的实现方式给出简要的解释和提示。

在 Transact-SQL 中，可使用两类注释符：

☑ ANSI 标准的注释符（--），用于单行注释；例如，下面 SQL 语句所加的注释：

```
USE pubs      --打开数据表
```

☑ 与 C 语言相同的程序注释符号，即“/*”“*/”。其中，“/*”用于注释文字的开头，“*/”用于注释文字的结尾，可在程序中标识多行文字为注释。例如，有多行注释的 SQL 语句如下：

```
USE student
DECLARE @songname char(10)
SELECT @songname=课程内容 FROM course WHERE 课程类别='艺术类'
PRINT @songname
/*打开 student 数据库，定义一个变量
把查询到的结果赋值给所定义的变量*/
```

把所选的行一次都注释的快捷键是 Shift+Ctrl+C；一次取消多行注释的快捷键是 Shift+Ctrl+R。

8.4.2　运算符（Operator）

运算符是一种符号，用来进行常量、变量或者列之间的数学运算和比较操作，它是 Transact-SQL 语言很重要的部分。运算符分为算术运算符、赋值运算符、比较运算符、逻辑运算符、位运算符、字符串连接运算符 6 种类型。

1. 算术运算符

算术运算符在两个表达式上执行数学运算，这两个表达式可以是数字数据类型分类的任何数据类型。算术运算符包括+（加）、-（减）、×（乘）、/（除）、%（取余）。

例如：5%3=2，3%5=3。

示例：求 2 对 5 取余。在查询编辑器窗口中运行的结果如图 8.5 所示。

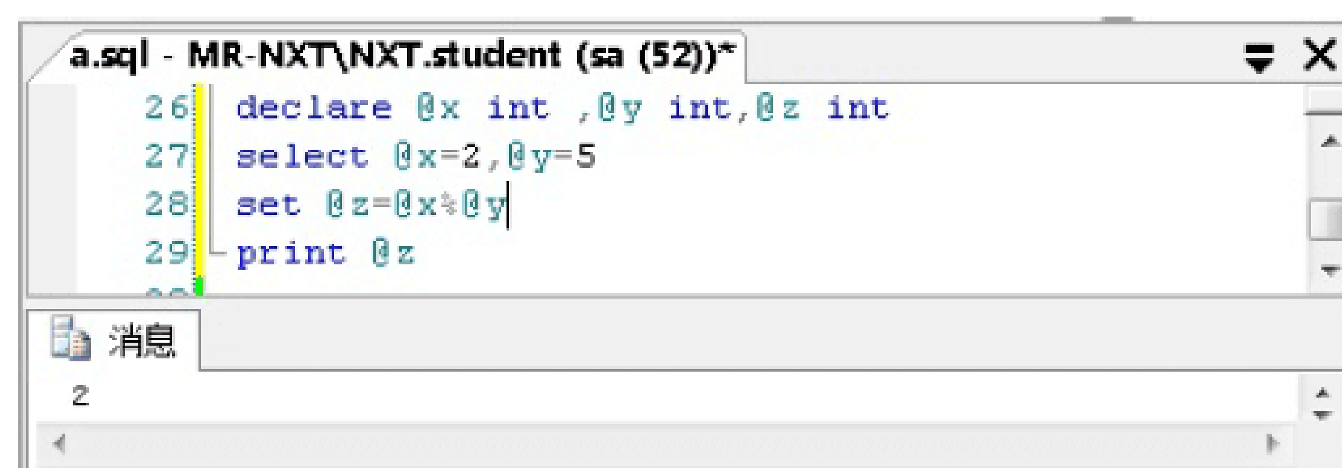


图 8.5　求 2%6 的结果

SQL 语句如下：

```
declare @x int ,@y int,@z int
select @x=2,@y=5
set @z=@x%@y
print @z
```



注意

取余运算两边的表达式必须是整型数据。

2．赋值运算符

Transact-SQL 有一个赋值运算符，即等号（=）。在下面的示例中，创建了@songname 变量。然后利用赋值运算符将@songname 设置成一个由表达式返回的值。

```
DECLARE @songname char(20)
SET @songname='loving'
```

还可以使用 SELECT 语句进行赋值，并输出该值。

```
DECLARE @songname char(20)
SELECT @songname ='loving'
print @songname
```

3．比较运算符

比较运算符测试两个表达式是否相同。除了 text、ntext 或 image 数据类型的表达式外，比较运算符可以用于所有的表达式。比较运算符包括>（大于）、<（小于）、=（等于）、>=（大于等于）、<=（小于等于）、!=（不等于）、!>（不大于）、!<（不小于），其中！=、!>、!<不是 ANSI 标准的运算符。

比较运算符的结果，布尔数据类型有 3 种值：TRUE、FALSE 及 UNKNOWN。那些返回布尔数据类型的表达式被称为布尔表达式。

和其他 SQL Server 数据类型不同，不能将布尔数据类型指定为表列或变量的数据类型，也不能在结果集中返回布尔数据类型。

例如：3>5=FALSE，6!=9=TRUE。

【例 8.05】　用查询语句搜索 pubs 数据库中的 titles 表，返回书的价格打了 8 折后仍大于 12 美元的书的书号、种类以及原价。（**实例位置：资源包\源码\06\8.05**）

SQL 语句如下：

```
use pubs
go
select title_id as 书号,type as 种类,price as 原价
from titles
where price-price*0.2>12
```

4．逻辑运算符

逻辑运算符对某个条件进行测试，以获得其真实情况。逻辑运算符和比较运算符一样，返回带有 TRUE 或 FALSE 值的布尔数据类型。SQL 支持的逻辑运算符如表 8.7 所示。

表 8.7　SQL 支持的逻辑运算符

运　算　符	行　　为
ALL	如果一个比较集中全部都是 TRUE，则值为 TRUE
AND	如果两个布尔表达式均为 TRUE，则值为 TRUE
ANY	如果一个比较集中任何一个为 TRUE，则值为 TRUE
BETWEEN	如果操作数是在某个范围内，则值为 TRUE
EXISTS	如果子查询包含任何行，则值为 TRUE
IN	如果操作数与一个表达式列表中的某个相等的话，则值为 TRUE
LIKE	如果操作数匹配某个模式的话，则值为 TRUE
NOT	对任何其他布尔运算符的值取反
OR	如果任何一个布尔表达式是 TRUE，则值为 TRUE
SOME	如果一个比较集中的某些为 TRUE 的话，则值为 TRUE

例如：8>5 and 3>2=TRUE。

【例 8.06】　在 student 表中，查询女生中年龄大于 21 岁的学生信息。在查询编辑器窗口中运行的结果如图 8.6 所示。（**实例位置：资源包\源码\06\8.06**）

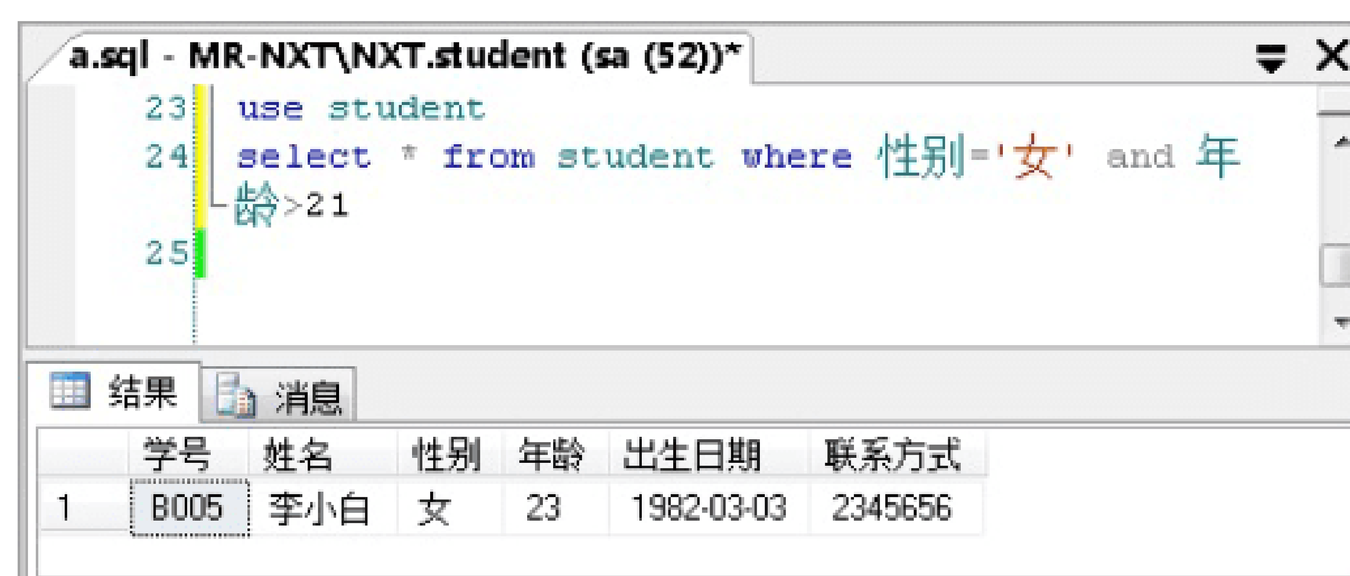


图 8.6　查询年龄大于 21 的女生信息

SQL 语句如下：

```
use student
select *
from student
where 性别='女' and 年龄>21
```

当 NOT、AND 和 OR 出现在同一表达式中，它们的优先级依次是 NOT、AND、OR。

例如：3>5 or 6>3 and not 6>4=FALSE。

先计算 not 6>4=FALSE，然后再计算 6>3 AND FALSE =FALSE，最后计算 3>5 or FALSE= FALSE。

5．位运算符

位运算符的操作数可以是整数数据类型或二进制串数据类型（image 数据类型除外）范畴的。SQL 支持的按位运算符如表 8.8 所示。

表 8.8　位运算符

运　算　符	说　　明
&	按位 AND
|	按位 OR
^	按位互斥 OR
~	按位 NOT

6．字符串连接运算符

字符串连接运算符（+）用于连接两个或两个以上的字符或二进制串、列名或者串和列的混合体，将一个串加入另一个串的末尾。

语法格式如下：

```
<expression1>+<expression2>
```

【例 8.07】　用“+”连接两个字符串。在查询编辑器窗口中运行的结果如图 8.7 所示。（**实例位置：资源包\源码\06\8.07**）

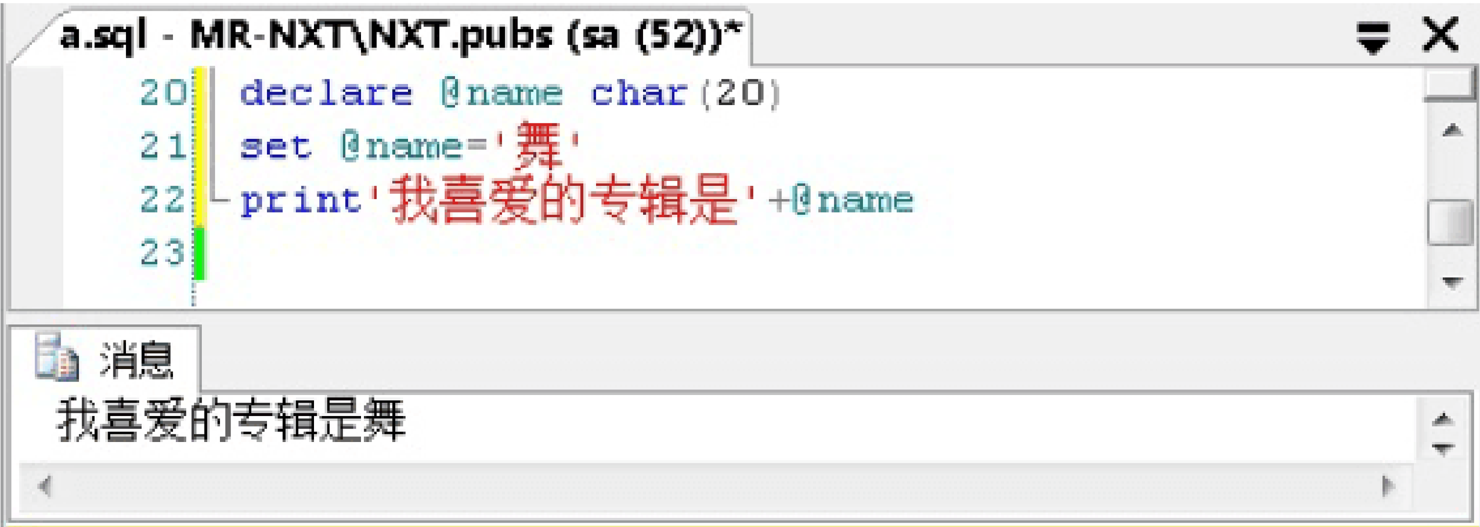


图 8.7　用“+”连接两个字符串

SQL 语句如下：

```
declare @name char(20)
set @name='舞'
print '我喜爱的专辑是'+@name
```

7．运算符优先级

当一个复杂表达式中包含多个运算符时，运算符的优先级决定了表达式计算和比较操作的先后顺序。运算符的优先级由高到低的顺序如下。

（1）+（正）、-（负）、~（位反）

（2）*（乘）、/（除）、%（取余）

（3）+（加）、+（字符串串联运算符）、-（减）

（4）=、>、<、>=、<=、<>、!=、!>、!<（比较运算符）

（5）^（按位异或）、&（按位与）、|（按位或）

（6）NOT

（7）AND

（8）ALL、ANY、BETWEEN、IN、LIKE、OR、SOME（逻辑运算符）

（9）=（赋值）

若表达式中含有相同优先级的运算符，则从左向右依次处理。还可以使用括号来提高运算的优先级，在括号中的表达式优先级最高。如果表达式有嵌套的括号，那么首先对嵌套最内层的表达式求值。例如：

```
DECLARE @num int
SET @num = 2 * (4 + (5 - 3))
```

先计算（5−3），然后再加 4，最后再和 2 相乘。

8.4.3　通配符（Wildcard）

在 SQL 中通常用 LIKE 关键字与通配符结合起来实现模式查询。其中 SQL 支持的通配符如表 8.9 所示。

表 8.9　SQL 支持的通配符的描述和示例

通　配　符	描　　述	示　　例
%	包含零个或更多字符的任意字符	'loving%'可以表示：'loving'、'loving you'、'loving? '
_（下画线）	任何单个字符	'loving_'可以表示：'lovingc'。后面只能再接一个字符
[ ]	指定范围（[a-f]）或集合（[abcdef]）中的任何单个字符	'[0-9]123'表示以 0～9 之间任意一个字符开头，以'123'结尾的字符
[^]	不属于指定范围（[a-f]）或集合（[abcdef]）的任何单个字符	'[^0-5]123'表示不以 0～5 之间任意一个字符开头，却以'123'结尾的字符

8.5　小　　结

本章介绍了 Transact-SQL 语法基础，常量、变量、注释符、运算符与通配符的运用，以及运算符的优先级和如何比较运算符等，都能使读者更好地理解所学的知识。

第 9 章

数据的查询

（视频讲解：32 分钟）

本章主要介绍针对数据表记录的各种查询以及对记录的操作，主要包括选择查询、数据汇总、基于多表的连接查询。通过本章的学习，读者可以应用各种查询对数据表中的记录进行访问。

学习摘要：

- 创建查询和测试查询
- 简单的 SELECT 查询
- 使用 WHERE 子句过滤数据
- 使用聚合函数
- 使用 GROUP BY 子句
- 使用 HAVING 子句
- 使用 JOIN 关键字连接多个数据表

9.1　创建查询和测试查询



1. 编写 SQL 语句

在 SQL Server 2014 中，用户可以在 SQL Server Manager Studio 中编写 SQL 语句操作数据库。例如，查询 course 表中的所有记录的操作步骤如下。

（1）选择“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 命令，启动 SQL Server Manager Studio。

（2）使用“Windows 身份验证”建立连接。

（3）单击“标准”工具栏上的“新建查询”按钮。

（4）输入如下 SQL 语句：

```
use student
select * from course
```

2. 测试 SQL 语句

在新建的查询编辑器窗口中输入 SQL 语句之后，为了查看语句是否有语法错误，需要对 SQL 语句进行测试。单击工具栏中的✓按钮或直接按 Ctrl+F5 快捷键，可以对当前的 SQL 语句进行测试，如果 SQL 语句准确无误，在代码区下方会显示“命令已成功完成”，否则显示错误信息提示。

3. 执行 SQL 语句

最后要执行 SQL 语句才能实现各种操作。单击工具栏上的执行(X)按钮或直接按 F5 键可以执行 SQL 语句。上面输入的 SQL 语句的执行结果如图 9.1 所示。

SQLQuery1.sql - MR....student (sa (52))*

```
use student
select * from course
```

结果　消息

	课程代号	课程名称	课程类别	课程内容
1	k01	喜爱的逻辑	艺术类	童年
2	k02	喜爱的逻辑	艺术类	童年2

图 9.1　显示 course 表的所有记录

9.2　选 择 查 询



9.2.1　简单的 SELECT 查询

SELECT 语句是从数据库中检索数据并查询，并将查询结果以表格的形式返回。

SELECT 语句的基本语法格式如下：

```
SELECT select_list
[INTO new_table_name]
FROM table_list
[WHERE search_condition]
[GROUP BY group_by_list]
[HAVING search_condition]
[ORDER BY order_list [ASC| DESC]]
```

参数说明如表 9.1 所示。

表 9.1　SELECT 语句的参数说明

设　置　值	描　　述
select_list	指定由查询返回的列，它是一个逗号分隔的表达式列表
INTO new_table_name	创建新表并将查询行从查询插入新表中。new_table_name 指定新表的名称
FROM table_list	指定从其中检索行的表，这些来源可能包括基表、视图和链接表；FROM 子句还可包含连接说明；FROM 子句还用在 DELETE 和 UPDATE 语句中以定义要修改的表
WHERE search_condition	WHERE 子句指定用于限制返回的行的搜索条件。WHERE 子句还用在 DELETE 和 UPDATE 语句中以定义目标表中要修改的行
GROUP BY group_by_list	GROUP BY 子句根据 group_by_list 列中的值将结果集分成组。例如，student 表在“性别”中有两个值。“GROUP BY 性别”子句将结果集分成两组，每组对应于性别的一个值
HAVING search_condition	HAVING 子句是指定组或聚合的搜索条件
ORDER BY order_list [ASC | DESC]	ORDER BY 子句定义结果集中的行排列的顺序。order_list 指定组成排序列表的结果集的列。ASC 和 DESC 关键字用于指定行是按升序还是按降序排序

1．选择所有字段

SELECT 语句后的第一个子句，即 SELECT 关键字开头的子句，用于选择进行显示的列。如果要显示数据表中所有列的值时，SELECT 子句后用星号（*）表示。

【例 9.01】　查询包含所有字段的记录。（**实例位置：资源包\源码\09\9.01**）

在 student 数据库中，查询 grade 表的所有记录，查询结果如图 9.2 所示。

	学号	课程代号	课程成绩	学期
1	B003	k03	90.3	1
2	B005	K02	93.2	2
3	NULL	NULL	NULL	NULL
4	B003	K03	98.3	1
5	B004	K04	87.9	2
6	B002	K02	88.4	2

图 9.2　查询显示 grade 表的内容

SQL 语句如下：

```
USE student
SELECT *
FROM grade
```

2．选择部分字段

在查询表时，很多时候只显示所需要的字段，这时在 SELECT 子句后分别列出各个字段名称就可以。

【例 9.02】　查询包含部分字段的记录。（**实例位置：资源包\源码\09\9.02**）

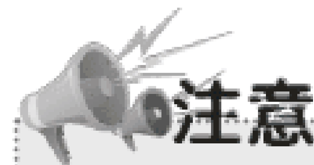
在 grade 表中，显示学号、课程成绩字段的信息。查询结果如图 9.3 所示。

	学号	课程成绩
1	B003	90.3
2	B005	93.2
3	NULL	NULL
4	B003	98.3
5	B004	87.9
6	B002	88.4

图 9.3　显示 grade 表的部分列

SQL 语句如下：

```
USE student
SELECT 学号,课程成绩
FROM grade
```


注意

各个列用逗号隔开，但逗号是英文状态下的逗号。且不要混淆 SELECT 子句和 SELECT 语句。

9.2.2　重新对列排序

对于表格比较小的，不用 ORDER BY 子句，查询结果会按照在表格中的顺序排列。但对于表格比较大的，则必须使用 ORDER BY 子句，方便查看查询结果。

ORDER BY 子句由关键字 ORDER BY 后跟一个用逗号分开的排序列表组成，语法格式如下：

```
[ORDER BY {order_by_expression [ASC | DESC]} [,...n]]
```

参数说明如表 9.2 所示。

表 9.2　ORDER BY 语句的参数说明

设　置　值	说　　明
order_by_expression	指定要排序的列。可以将排序列指定为列名、列的别名（可由表名或视图名限定）和表达式，或者指定为代表选择列表内的名称、别名或表达式的位置的负整数。可指定多个排序列。ORDER BY 子句中的排序列序列定义排序结果集的结构
ORDER BY	子句可包括未出现在此选择列表中的项目。然而，如果指定 SELECT DISTINCT，或者如果 SELECT 语句包含 UNION 运算符，则排序列必定出现在选择列表中。此外，当 SELECT 语句包含 UNION 运算符时，列名或列的别名必须是在第一选择列表内指定的列名或列的别名
ASC	指定按递增顺序，从低到高对指定列中的值进行排序。默认就是递增顺序
DESC	指定按递减顺序，从高到低对指定列中的值进行排序

1．单级排序

排序的关键字是 ORDER BY，默认状态下是升序，关键字是 ASC。可以按照某一个字段排序，排序的字段是数值型，也可以是字符型、日期和时间型。

【例 9.03】　按照某一个字段进行排序。(**实例位置：资源包\源码\09\9.03**)

在 tb_basicMessage 表中，按照 age 升序排序。查询结果如图 9.4 所示。

	id	name	age	sex	dept	headship
1	24	金额	20	女	4	7
2	25	cdd	24	女	3	6
3	27	———	25	男	2	3
4	7	小陈	27	男	1	1
5	8	小葛	29	男	1	1
6	16	张三	30	男	1	5
7	23	小开	30	男	4	4

图 9.4　tb_basicMessage 表按照 age 升序排序

SQL 语句如下：

```
USE db_supermarket
SELECT *
FROM tb_basicMessage
ORDER BY age
```

查询结果以降序排序，必须在列名后指定关键字的 DESC。

例如，在 tb_basicMessage 表中按照 age 降序排序。SQL 语句如下：

```
USE student
SELECT * FROM tb_basicMessage ORDER BY age DESC
```

2．多级排序

按照一列进行排序后，如果该列有重复的记录值，则重复记录值这部分就没有进行有效的排序，这就需要再附加一个字段，作为第二次排序的标准，对没有排序的记录进行再排序。

【例 9.04】　按照多个字段进行排序。(**实例位置：资源包\源码\09\9.04**)

在 grade 表中，按照学生的“学期”降序排列，然后再按照“课程成绩”升序排序。查询结果如图 9.5 所示。

	学号	课程代号	课程成绩	学期
1	B004	K04	87.9	2
2	B002	K02	88.4	2
3	B005	K02	93.2	2
4	B003	k03	90.3	1
5	B001	K01	96.7	1
6	B003	K03	98.3	1

图 9.5　grade 表按照多级字段排序

SQL 语句如下：

```
USE student
SELECT *
```

```
FROM grade
ORDER BY 学期 DESC,课程成绩
```

当排序字段是字符类型时，将按照字符数据中字母或汉字的拼音在字典中的顺序排序，先比较第 1 个字母在字典中的顺序，位置在前的表示该字符串小于后面的字符串，若第 1 个字符相同，则继续比较第 2 个字母，直至得出比较结果。

例如，在 course 表中先按照“课程类别”升序排列，再按照“课程内容”降序排列。SQL 语句如下：

```
USE student
SELECT * FROM course ORDERY BY 课程类别 ASC,课程内容 DESC
```

9.2.3　使用运算符或函数进行列计算

某些查询要求在字段上带表达式进行查询，关于表达式中运算符和函数部分请参考 Transact-SQL 语法部分。

带表达式的查询语法如下：

```
SELECT 表达式 1,表达式 2,字段 1,字段 2,...from 数据表名
```

【例 9.05】　使用运算符进行列计算。（**实例位置：资源包\源码\09\9.05**）

新的一年开始学生的年龄都长了一岁，查询代码如下：

```
SELECT 学号,姓名,年龄=年龄+1 FROM tb_stu
```

查询结果如图 9.6 所示。

	学号	姓名	年龄
1	2	张二	24
2	3	张三	24
3	4	张四	24
4	5	张五	22
5	1	张一	21

图 9.6　表达式查询

9.2.4　利用 WHERE 参数过滤数据

WHERE 子句是用来选取需要检索的记录。因为一个表通常会有数千条记录，在查询结果中，用户仅需其中的一部分记录，这时需要使用 WHERE 子句指定一系列的查询条件。

WHERE 子句的基本语法格式如下：

```
SELECT<字段列表>
FROM<表名>
WHERE<条件表达式>
```

为了实现许多不同种类的查询，WHERE 子句提供了丰富的查询条件，下面总结了 5 个基本的查询条件。

（1）比较查询条件（=、<>、<、>等）。

（2）范围查询条件（BETWEEN、NOT BETWEEN）。

（3）列表查询条件（IN、NOT IN）。

（4）模糊查询（LIKE、NOT LIKE）。

（5）复合查询条件（AND、OR、NOT）。

1．比较查询条件

比较查询条件由比较运算符连接表达式组成，系统将根据该查询条件的真假来决定某一条记录是否满足该查询条件，只有满足该查询条件的记录才会出现在最终的结果集中。SQL Server 比较运算符如表 9.3 所示。

表 9.3　比较运算符

运　算　符	说　　明	运　算　符	说　　明
=	等于	<=	小于等于
>	大于	!>	不大于
<	小于	!<	不小于
>=	大于等于	<>或!=	不等于

【例 9.06】　使用运算符进行比较查询。（**实例位置：资源包\源码\09\9.06**）

在 grade 表中，查询“课程成绩”大于 90 分的，查询结果如图 9.7 所示。

	学号	课程代号	课程成绩	学期
1	B003	k03	90.3	1
2	B005	K02	93.2	2
3	B003	K03	98.3	1
4	B001	K01	96.7	1

图 9.7　查询 grade 表中课程成绩大于 90 分的信息

SQL 语句如下：

```
USE student
SELECT *
FROM grade
WHERE 课程成绩>90
```

例如，在 grade 表中查询“课程成绩”小于等于 90 分的，SQL 语句如下：

```
USE student
SELECT * FROM grade WHERE 课程成绩<=90
```

例如，在 student 表中查询“年龄”范围为 20～22 岁（包括 20 和 22）的所有学生。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 年龄>=20 AND 年龄<=22
```

例如，在 student 表中查询“年龄”不大于 20～22 岁的所有学生。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 年龄<20 OR 年龄>22
```

例如，在 student 表中查询“年龄”不小于 20 岁的所有学生。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 年龄 !<20
```

换一种写法。查询“年龄”不小于 20 岁的所有学生。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 年龄>=20
```

例如，在 student 表中查询年龄不等于 20 岁的所有学生。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 年龄!=20
```


注意

搜索满足条件的记录行，要比消除所有不满足条件的记录行快得多，所以，将否定的 WHERE 条件改写为肯定的条件将会提高性能，这是一个必须记住的准则。

2. 范围查询条件

使用范围条件进行查询，是当需要返回某一个数据值是否位于两个给定值之间，通常使用 BETWEEN…AND 和 NOT…BETWEEN…AND 来指定范围条件。

使用 BETWEEN…AND 查询条件时，指定的第 1 个值必须小于第 2 个值。因为 BETWEEN…AND 实质是查询条件“大于等于第 1 个值，并且小于等于第 2 个值”的简写形式。即 BETWEEN…AND 要包括两端的值，等价于比较运算符（>=…<=）。

【例 9.07】 使用 BETWEEN…AND 语句进行范围查询。（**实例位置：资源包\源码\09\9.07**）

在 grade 表中，显示年龄范围为 20～21 岁的学生信息。查询结果如图 9.8 所示

	Sno	Sname	Sex	年龄
1	201109001	李羽凡	男	20

图 9.8　显示 grade 表中年龄范围为 20～21 岁的学生信息

SQL 语句如下：

```
USE student
SELECT *
FROM student
WHERE 年龄 BETWEEN 20 AND 21
```

上述 SQL 也可以用>=…<=符号来改写。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 年龄>=20 AND 年龄<=21
```

而 NOT…BETWEEN…AND 语句返回某个数据值在两个指定值范围以外，但并不包括两个指定的值。

【例 9.08】 使用 NOT…BETWEEN…AND 语句进行范围查询。（**实例位置：资源包\源码\09\9.08**）

在 student 表中，显示年龄不在 20～21 岁的学生信息。查询结果如图 9.9 所示。

	Sno	Sname	Sex	年龄
1	201109008	李艳丽	女	25
2	201109003	聂乐乐	女	23
3	201109018	触发器	男	23
4	201109002	王嘟嘟	女	22

图 9.9　显示 grade 表中年龄不在 20～21 岁的学生信息

SQL 语句如下：

```
USE student
SELECT *
FROM student
WHERE 年龄 NOT BETWEEN 20 AND 21
```

3．列表查询条件

当测试一个数据值是否匹配一组目标值中的一个时，通常使用 IN 关键字来指定列表搜索条件。IN 关键字的格式是 IN(目标值 1,目标值 2,目标值 3,…)，目标值的项目之间必须使用逗号分隔，并且括在括号中。

【例 9.09】　使用 IN 关键字进行列表查询。（**实例位置：资源包\源码\09\9.09**）

在 course 表中，查询“课程代号”是 k01、k03、k04 的课程信息。查询结果如图 9.10 所示。

	课程代号	课程名称	课程类别	课程内容
1	k01	喜爱的逻辑	艺术类	童年
2	k03	个人单曲	歌曲类	舞
3	k04	经典歌曲	歌曲类	冬天快乐

图 9.10　查询“课程代号”是 k01、k03、k04 的课程信息

SQL 语句如下：

```
USE student
SELECT *
FROM course
WHERE 课程代号 IN ('k01','k03', 'k04')
```

IN 运算符可以与 NOT 配合使用排除特定的行。测试一个数据值是否不匹配任何目标值。

【例 9.10】　使用 NOT IN 关键字进行列表查询。（**实例位置：资源包\源码\09\9.10**）

在 course 表中，查询“课程代号”不是 k01、k03 和 k04 的课程信息。查询结果如图 9.11 所示。

	课程代号	课程名称	课程类别	课程内容
1	k02	喜爱的逻辑	艺术类	童年2

图 9.11　查询“课程代号”不是 k01、k03、k04 的课程信息

SQL 语句如下：

```
USE student
SELECT *
FROMcourse
WHERE 课程代号 NOT IN('k01','k03', 'k04')
```

4．模糊查询

有时用户对查询数据表中的数据了解得不全面，如不能确定所要查询人的姓名只知道姓李，查询某个人的联系方式只知道是以 3451 结尾等，这时需要使用 LIKE 进行模糊查询。LIKE 关键字需要使用通配符在字符串内查找指定的模式，所以读者需要了解通配符及其含义。通配符的含义如表 9.4 所示。

表 9.4　LIKE 关键字中的通配符及其含义

通　配　符	说　　明
%	由零个或更多字符组成的任意字符串
_	任意单个字符
[]	用于指定范围，例如，[A-F]表示 A～F 范围内的任何单个字符
[^]	表示指定范围之外的，例如，[^A-F]范围以外的任何单个字符

（1）%通配符

%通配符能匹配 0 个或更多个字符的任意长度的字符串。

【例 9.11】　使用%通配符进行模糊查询。（**实例位置：资源包\源码\09\9.11**）

在 student 表中，查询姓“李”的学生信息。查询结果如图 9.12 所示。

	学号	姓名	性别	年龄
1	201109008	李艳丽	女	25
2	201109001	李羽凡	男	20

图 9.12　student 表中查询姓“李”的学生信息

SQL 语句如下：

```
USE student
SELECT *
FROM student
WHERE 姓名 LIKE '李%'
```

在 SQL Server 语句中，可以在查询条件的任意位置放置一个%符号来代表任意长度的字符串。在设置查询条件时，也可以放置两个%，但最好不要连续出现两个%符号。

例如，在 student 表中，查询姓“李”并且联系方式是以 2 打头的学生信息。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 姓名 LIKE '李%' AND 联系方式 LIKE '2%'
```

（2）_通配符

_号表示任意单个字符，该符号只能匹配一个字符，利用_号可以作为通配符组成匹配模式进行查询。

【例 9.12】　使用_通配符进行模糊查询。（**实例位置：资源包\源码\09\9.12**）

在 student 表中，查询姓“刘”并且名字只有两个字的学生信息。查询结果如图 9.13 所示。

	学号	姓名	性别	年龄	出生日期	联系方式
1	201109004	刘月	女	20	1986-01-03	82345

图 9.13　在 student 表中查询姓“刘”并且名字是两个字的学生信息

SQL 语句如下：

```
USE student
SELECT *
FROM student
WHERE 姓名 LIKE '刘_'
```

_符号可以放在查询条件的任意位置，但只能代表一个字符。

例如，在 student 表中，查询姓“李”并且末尾字是“丽”的学生信息。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 姓名 LIKE '李_丽'
```

（3）[]通配符

在模糊查询中可以使用[]符号来查询一定范围内的数据。[]符号用于表示一定范围内的任意单个字符，它包括两端数据。

【例 9.13】 使用[]通配符进行模糊查询。（**实例位置：资源包\源码\09\9.13**）

在 student 表中，查询联系方式以 3451 结尾并且开头数字位于 1～5 之间的学生信息。查询结果如图 9.14 所示。

	学号	姓名	性别	年龄	出生日期	联系方式
1	201109008	李艳丽	女	25	1985-03-03	13451
2	201109003	聂乐乐	女	23	1984-03-10	23451
3	201109001	李羽凡	男	20	1982-03-03	23451

图 9.14　在 student 表中查询联系方式以 3451 结尾的学生

SQL 语句如下：

```
USE student
SELECT *
FROM student
WHERE 联系方式 LIKE '[1-5]3451'
```

例如，在 grade 表中，查询学号是 B001～B003 之间的学生成绩信息。SQL 语句如下：

```
USE student
SELECT * FROM grade WHERE 学号 LIKE 'B00[1-3]'
```

（4）[^]通配符

在模糊查询中可以使用[^]符号来查询不在指定范围内的数据。[^]符号用于表示不在某范围内的任意单个字符，它包括两端数据。

【例 9.14】 使用[^]通配符进行模糊查询。（**实例位置：资源包\源码\09\9.14**）

在 student 表中，查询联系方式以 3451 结尾，但不以 2 开头的学生信息。查询结果如图 9.15 所示。

	学号	姓名	性别	年龄	出生日期	联系方式
1	201109008	李艳丽	女	25	1985-03-03	13451

图 9.15　在 student 表中查询联系方式以 3451 结尾但不以 2 开头的学生信息

SQL 语句如下：

```
USE student
SELECT *
```

```
FROM student
WHERE 联系方式 LIKE '[^2]3451'
```

NOT LIKE 的含义与 LIKE 关键字正好相反，查询结果将返回不符合匹配模式查询。

例如，查询不姓“李”的学生信息。SQL 语句如下：

```
SELECT * FROM student WHERE 姓名 NOT LIKE '李%'
```

例如，查询除了名字是两个字并且姓“李”的其他学生信息。SQL 语句如下：

```
SELECT * FROM student WHERE 姓名 NOT LIKE '李_'
```

例如，查询除了电话号码以 3451 结尾并且开头数字位于 1～5 之间的其他学生信息。SQL 语句如下：

```
SELECT * FROM student WHERE 联系方式 NOT LIKE '[1-5]3451'
```

例如，查询电话号码不符合如下条件的学生信息，这些条件是电话号码以 3451 结尾，但不以 2 开头的。SQL 语句如下：

```
SELECT * FROM student WHERE 联系方式 NOT LIKE '[^2]3451'
```

5．复合查询条件

很多情况下，在 WHERE 子句中仅仅使用一个条件不能准确地从表中检索到需要的数据，这里就需要使用逻辑运算符 AND、OR 和 NOT。使用逻辑运算符时，遵循的指导原则如下。

（1）使用 AND 返回满足所有条件的行。

（2）使用 OR 返回满足任一条件的行。

（3）使用 NOT 返回不满足表达式的行。

例如，用 OR 进行查询。查询学号是 B001 或者是 B003 的学生信息。SQL 语句如下：

```
USE student
SELECT * FROM student WHERE 学号='B001' OR 学号='B003'
```

例如，用 AND 进行查询。根据姓名和密码查询用户。SQL 语句如下：

```
USE db_supermarket
SELECT * FROM tb_users WHERE username='mr' AND password='mrsoft'
```

就像数据运算符乘和除一样，它们之间是具有优先级顺序的：NOT 优先级最高，AND 次之，OR 的优先级最低。下面用 AND 和 OR 结合进行查询。

【例 9.15】　使用 AND 和 OR 结合进行查询。（**实例位置：资源包\源码\09\9.15**）

在 student 表中，要查询年龄大于 21 岁女生或者年龄大于等于 19 岁的男生信息。查询结果如图 9.16 所示。

	学号	姓名	性别	年龄	出生日期	联系方式
1	201109008	李艳丽	女	25	1985-03-03	13451
2	201109003	聂乐乐	女	23	1984-03-10	23451
3	201109018	触发器	男	23	1986-01-01	52345
4	201109002	王嘟嘟	女	22	1984-03-10	62345
5	201109001	李羽凡	男	20	1982-03-03	23451

图 9.16　复合搜索

SQL 语句如下：

```
USE student
SELECT *
FROM student
WHERE 年龄> 21 AND 性别='女' OR 年龄>=19 AND 性别='男'
```

使用逻辑关键字 AND、OR、NOT 和括号把搜索条件分组，可以构建非常复杂的搜索条件。

例如，在 student 表中，查询年龄大于 20 岁的女生或者年龄大于 22 岁的男生，并且电话号码都是 23451 的学生信息。在查询编辑器窗口中输入的 SQL 语句如下：

```
USE student
SELECT * FROM student WHERE (年龄>20 AND 性别='女' OR 年龄>22 AND 性别='男') AND 联系方式 =
'23451'
```

9.2.5 消除重复记录

DISTINCT 关键字主要用来从 SELECT 语句的结果集中去掉重复的记录。如果用户没有指定 DISTINCT 关键字，那么系统将返回所有符合条件的记录组成结果集，其中包括重复的记录。

【例 9.16】 使用 DISTINCT 关键字消除重复记录。（**实例位置：资源包\源码\09\9.16**）

在 course 表中，显示共有几种“课程类别”。查询结果如图 9.17 所示。

	课程类别
1	歌曲类
2	计算机类
3	艺术类

图 9.17 显示 course 表中的课程类别

SQL 语句如下：

```
USE student
SELECT DISTINCT 课程类别
FROM course
```

对多个列使用 DISTINCT 关键字时，查询结果只显示每个有效组合的一个例子。即结果表中没有完全相同的两行。

例如，在 grade 表中，显示“学号”和“课程代号”的不同值。SQL 语句如下：

```
USE student
SELECT DISTINCT 学号,课程代号
FROM grade
```



视频讲解

9.3 数据汇总

9.3.1 使用聚合函数

SQL 提供一组聚合函数，它们能够对整个数据集合进行计算，将一组原始数据转换为有用的信息，以便用户使用。例如，求成绩表中的总成绩、学生表中平均年龄等。

SQL 的聚合函数如表 9.5 所示。

表 9.5　聚合函数

聚 合 函 数	支持的数据类型	功 能 描 述
SUM()	数字	对指定列中的所有非空值求和
AVG()	数字	对指定列中的所有非空值求平均值
MIN()	数字、字符、日期	返回指定列中的最小数字、最小的字符串和最早的日期时间
MAX()	数字、字符、日期	返回指定列中的最大数字、最大的字符串和最近的日期时间
COUNT([DISTINCT] *)	任意基于行的数据类型	统计结果集中全部记录行的数量。最多可达 2147483647 行
COUNT_BIG([DISTINCT] *)	任意基于行的数据类型	类似于 COUNT 函数，但因其返回值使用了 BIGINT 数据类型，所以最多可以统计 $2^{63}-1$ 行

下面用聚集函数分别举例。

例如，在 grade 表中，求所有的课程成绩的总和，SQL 语句如下：

```
USE student
SELECT SUM(课程成绩) FROM grade
```

例如，在 student 表中，求所有学生的平均年龄，SQL 语句如下：

```
USE student
SELECT AVG(年龄) FROM student
```

例如，在 student 表中，查询最早出生的学生，SQL 语句如下：

```
USE student
SELECT MIN(出生日期) FROM student
```

例如，在 grade 表中，查询课程成绩最高的学生信息，SQL 语句如下：

```
USE student
SELECT MAX(课程成绩) FROM grade
```

例如，在 student 表中，求所有女生的人数，SQL 语句如下：

```
USE student
SELECT COUNT(性别) FROM student WHERE  性别='女'
```

使用 COUNT(*)可以求整个表所有的记录数。

例如，求 student 表中所有的记录数，SQL 语句如下：

```
USE student
SELECT COUNT(*) FROM student
```

9.3.2　使用 GROUP BY 子句

GROUP BY 子句可以将表的行划分为不同的组。分别总结每个组，这样就可以控制想要看见的详细信息的级别。例如，按照学生的性别分组、按照不同的学期分组等。

使用 GROUP BY 子句的注意事项。

（1）在 SELECT 子句的字段列表中，除了聚集函数外，其他所出现的字段一定要在 GROUP BY 子句中有定义才行。如 GROUP BY A,B，那么 SELECT SUM(A),C 就有问题，因为 C 不在 GROUP BY 中，但是 SUM(A)还是可以的。

（2）SELECT 子句的字段列表中不一定要有聚集函数，但至少要用到 GROUP BY 子句列表中的一个项目。如 GROUP BY A,B,C，则 SELECT A 是可以的。

（3）在 SQL Server 中 text、ntext 和 image 数据类型的字段不能作为 GROUP BY 子句的分组依据。

（4）GROUP BY 子句不能使用字段别名。

1．按单列分组

GROUP BY 子句可以基于指定某一列的值将数据集合划分为多个分组，同一组内所有记录在分组属性上具有相同值。

【例 9.17】　使用 GROUP BY 子句按单列分组。（**实例位置：资源包\源码\09\9.17**）

把 student 表按照“性别”这个单列进行分组。查询结果如图 9.18 所示。

	性别
1	男
2	女

图 9.18　student 表按照“性别”分组

SQL 语句如下：

```
USE student
SELECT 性别
FROM student
GROUP BY 性别
```

重复前面介绍的注意事项：在 SELECT 子句的字段列表中，除了聚集函数外，其他所出现的字段一定要在 GROUP BY 子句中有定义才行。

例如，由于下列查询中“姓名”列即不包含在 GROUP BY 子句中，也不包含在分组函数中，所以是错误的。错误的 SQL 语句如下：

```
USE student SELECT 姓名,性别 FROM student GROUP BY 性别
```

2．按多列分组

GROUP BY 子句可以基于指定多列的值将数据集合划分为多个分组。

【例 9.18】　使用 GROUP BY 子句按多列分组。（**实例位置：资源包\源码\09\9.18**）

在 student 表中，按照“性别”和“年龄”列进行分组。查询结果如图 9.19 所示。

	性别	年龄
1	男	20
2	男	23
3	女	20
4	女	22
5	女	23
6	女	25

图 9.19　student 表按多列分组

SQL 语句如下：

```
USE student
SELECT 性别,年龄
FROM student
GROUP BY 性别,年龄
```

在 student 表中，首先按照性别分组，然后再按照年龄分组。

9.3.3　使用 HAVING 子句

分组之前的条件要用 WHERE 关键字，而分组之后的条件要使用关键字 HAVING 子句。

【例 9.19】　使用 HAVING 子句分组查询。（**实例位置：资源包\源码\09\9.19**）

在 student 表中，先按“性别”分组求出平均年龄，然后筛选出平均年龄大于 20 岁的学生信息。查询结果如图 9.20 所示。

	(无列名)	性别
1	21	男
2	22	女

图 9.20　student 表用 HAVING 筛选结果

SQL 语句如下：

```
USE student
SELECT AVG(年龄), 性别
FROM student
GROUP BY 性别
HAVING AVG(年龄)>20
```

9.4　基于多表的连接查询



视频讲解

9.4.1　连接谓词

JOIN 是一种将两个表连接在一起的连接谓词。连接条件可在 FROM 或 WHERE 子句中指定，建议在 FROM 子句中指定连接条件。

9.4.2　以 JOIN 关键字指定的连接

使用 JOIN 关键字可以进行交叉连接、内连接和外连接。

1. 交叉连接

交叉连接是两个表的笛卡儿积的另一个名称。笛卡儿积就是两个表的交叉乘积，即两个表的记录

进行交叉组合，如图 9.21 所示。

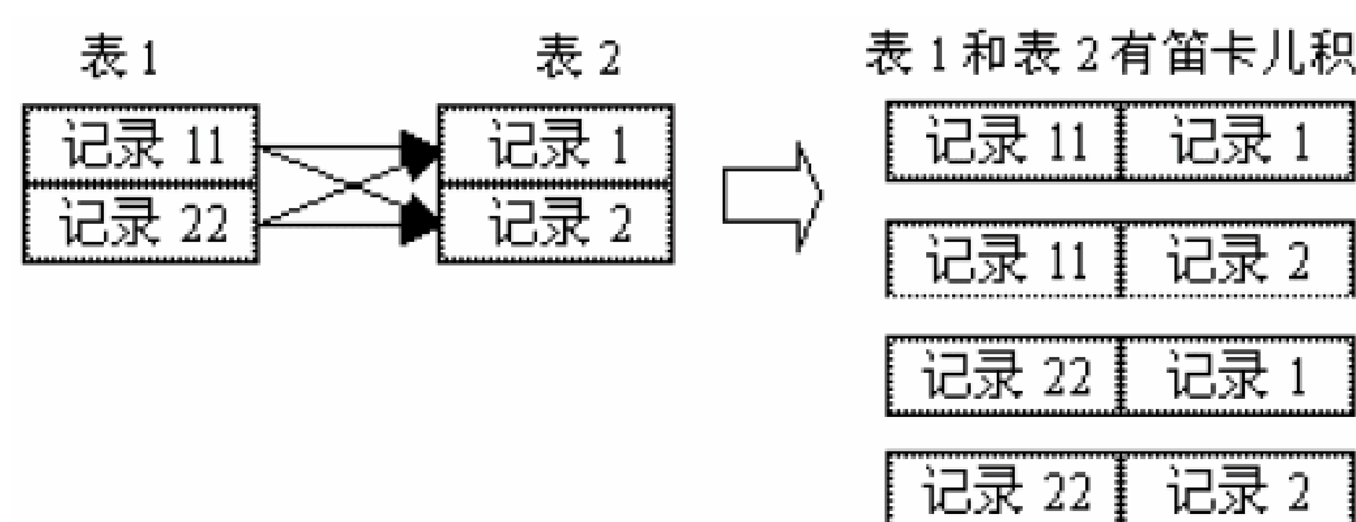


图 9.21　两个表的笛卡儿积示意图

交叉连接的语法格式如下：

```
SELECT fieldlist
FROM table1
CROSS JOIN table2
```

其中忽略 ON 条件的方法来创建交叉连接。

2．内连接

内连接也叫连接，是最早的一种连接，还被称为普通连接或自然连接。内连接是从结果中删除其他被连接表中没有匹配行的所有行，所以内连接可能会丢失信息。

内连接的语法格式如下：

```
SELECT fieldlist
FROM table1 [INNER] JOIN table2
ON table1.column=table2.column
```

一个表中的行和与另外一个表中的行匹配连接。表中的数据决定了如何对这些行进行组合。从每一个表中选取一行。

3．外连接

外连接则扩充了内连接的功能，会把内连接中删除原表中的一些保留下来，由于保留下来的行不同，把外连接分为左外连接、右外连接和全外连接 3 种连接。

（1）左外连接

左外连接保留了第 1 个表的所有行，但只包含第 2 个表与第 1 个表匹配的行。第 2 个表相应的空行被放入 NULL 值。

左外连接的语法格式如下：

```
USE student
SELECT fieldlist
FROM table1 LEFT JOIN table2
ON table1.column= table2.column
```

【例 9.20】　使用 LEFT JOIN…ON 关键字进行左外连接。（**实例位置：资源包\源码\09\9.20**）

把 student 表和 grade 表左外连接。第 1 个表 student 有不满足连接条件的。查询结果如图 9.22 所示。

	学号	姓名	性别	年龄	出生日期	联系方式	学号	课程代号	课程成绩	学期
1	B001	李艳丽	女	25	1985-03-03	13451	B001	K01	96.7	1
2	B002	聂乐乐	女	23	1984-03-10	23451	B002	K02	88.4	2
3	B003	触发器	男	23	1986-01-01	52345	B003	k03	90.3	1
4	B003	触发器	男	23	1986-01-01	52345	B003	K03	98.3	1
5	B004	王嘟嘟	女	22	1984-03-10	62345	B004	K04	87.9	2
6	B005	李羽凡	男	20	1982-03-03	23451	B005	K02	93.2	2
7	B006	刘月	女	20	1986-01-03	82345	NULL	NULL	NULL	NULL

图 9.22　student 表和 grade 表左外连接

SQL 语句如下：

```
USE student
SELECT   *
FROM student LEFT JOIN grade
ON student.学号=grade.学号
```

（2）右外连接

右外连接保留了第 2 个表的所有行，但只包含第 1 表与第 2 个表匹配的行。第 1 个表相应的空行被放入 NULL 值。

右外连接的语法格式如下：

```
USE student
SELECT fieldlist
FROM table1 RIGHT JOIN table2
ON table1.column=table2.column
```

【例 9.21】　使用 RIGHT JOIN…ON 关键字进行右外连接。（**实例位置：资源包\源码\09\9.21**）

把 grade 表和 course 表右外连接。第 2 个表 course 有不满足连接条件的行。查询结果如图 9.23 所示。

	学号	课程代号	课程成绩	学期	课程代号	课程名称	课程类别	课程内容
1	B001	K01	96.7	1	k01	喜爱的逻辑	艺术类	童年
2	B005	K02	93.2	2	k02	喜爱的逻辑	艺术类	童年2
3	B002	K02	88.4	2	k02	喜爱的逻辑	艺术类	童年2
4	B003	k03	90.3	1	k03	个人单曲	歌曲类	舞
5	B003	K03	98.3	1	k03	个人单曲	歌曲类	舞
6	B004	K04	87.9	2	k04	经典歌曲	歌曲类	冬天快乐
7	NULL	NULL	NULL	NULL	k06	数据结构	计算机类	查询

图 9.23　grade 表和 course 表右外连接

SQL 语句如下：

```
USE student
SELECT *
FROM grade RIGHT JOIN course
ON course.课程代号=grade.课程代号
```

（3）全外连接

全外连接会把两个表所有的行都显示在结果表中，并尽可能多地匹配数据和连接条件。

全外连接的语法格式如下：

```
USE student
SELECT fieldlist
```

```
FROM table1 FULL JOIN table2
ON table1.column=table2.column
```

【例 9.22】 使用 JOIN 关键字进行全外连接。（**实例位置：资源包\源码\09\9.22**）

把 grade 表和 course 表实现全连接。两个表都有不满足连接条件的。查询结果如图 9.24 所示。

	学号	课程代号	课程成绩	学期	课程代号	课程名称	课程类别	课程内容
1	B003	k03	90.3	1	k03	个人单曲	歌曲类	舞
2	B005	K02	93.2	2	k02	喜爱的逻辑	艺术类	童年2
3	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	B003	K03	98.3	1	k03	个人单曲	歌曲类	舞
5	B004	K04	87.9	2	k04	经典歌曲	歌曲类	冬天快乐
6	B002	K02	88.4	2	k02	喜爱的逻辑	艺术类	童年2
7	B001	K01	96.7	1	k01	喜爱的逻辑	艺术类	童年
8	NULL	NULL	NULL	NULL	k06	数据结构	计算机类	查询

图 9.24　course 表和 grade 表全外连接

SQL 语句如下：

```
USE student
SELECT *
FROM grade FULL JOIN course
ON course.课程代号=grade.课程代号
```

9.5　小　　结

本章介绍了如何在 SQL Server 2014 中编写、测试和执行 SQL 语句。读者应熟练掌握选择查询、分组查询，能根据实际的要求编写 SQL 查询语句的操作。

第10章

子查询与嵌套查询

（视频讲解：11分钟）

在使用SELECT语句检索数据时，可以使用WHERE子句指定用于限制返回的行的搜索条件，GROUP BY子句将结果集分成组，ORDER BY子句定义结果集中的行排列的顺序。使用这些子句可以方便地查询表中的数据。但是，当由WHERE子句指定的搜索条件指向另一张表时，就需要使用子查询或嵌套查询。在本章将详细地介绍什么是子查询、嵌套查询以及如何进行嵌套查询。

学习摘要：

- **子查询的概念**
- **嵌套查询的概念**
- **简单的嵌套查询**
- **带关键字的嵌套查询**



10.1 子查询概述

子查询是一个嵌套在 SELECT、INSERT、UPDATE 或 DELETE 语句或其他子查询中的查询。任何允许使用表达式的地方都可以使用子查询。

10.1.1 子查询语法

子查询语法格式如下：

```
SELECT [ALL | DISTINCT]<select item list>
FROM <table list>
[WHERE<search condition>]
[GROUP BY <group item list>
[HAVING <group by search conditoon>]]
```

10.1.2 语法规则

（1）子查询的 SELECT 查询总使用圆括号括起来。
（2）不能包括 COMPUTE 或 FOR BROWSE 子句。
（3）如果同时指定 TOP 子句，则可能只包括 ORDER BY 子句。
（4）子查询最多可以嵌套 32 层，个别查询可能会不支持 32 层嵌套。
（5）任何可以使用表达式的地方都可以使用子查询，只要它返回的是单个值。
（6）如果某个表只出现在子查询中而不出现在外部查询中，那么该表中的列就无法包含在输出中。

10.1.3 语法格式

（1）WHERE 查询表达式 [NOT] IN（子查询）。
（2）WHERE 查询表达式 比较运算符 [ANY | ALL]（子查询）。
（3）WHERE [NOT] EXISTS（子查询）。

10.2 嵌套查询概述

嵌套查询是指将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询。

嵌套查询中上层的查询块称为外侧查询或父查询，下层查询块称为内层查询或子查询。SQL 语言允许多层嵌套，但是在子查询中不允许出现 ORDER BY 子句，ORDER BY 子句只能用在最外层的查询块中。

嵌套查询的处理方法是：先处理最内侧的子查询，然后一层一层向上处理，直到最外层的查询块。

10.3　简单的嵌套查询

嵌套查询中的内层子查询通常作为搜索条件的一部分呈现在 WHERE 或 HAVING 子句中。例如，把一个表达式的值和一个由子查询生成的一个值相比较，这个测试类似于简单比较测试。

子查询比较测试用到的运算符包括=、<>、<、>、<=、>=。子查询比较测试把一个表达式的值和由子查询产生的一个值进行比较，返回比较结果为 TRUE 的记录。

【例 10.01】 Student 表中存储的是学生的基本信息，SC 表中存储的是学生的成绩（Grade）信息，使用嵌套查询，查询在 Student 表中，Grade>90 分的学生信息，SQL 语句及运行结果如图 10.1 所示。**（实例位置：资源包\源码\10\10.01）**

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sno = (SELECT Sno FROM SC WHERE Grade > 90)
```

这里给出本节中用到的 SC 表、Student 表和 Course 表的信息，如图 10.2 所示。

```
2 SELECT * FROM Student
3 WHERE Sno = (SELECT Sno FROM SC WHERE Grade > 90)
```

结果　消息

	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	21

图 10.1　查询成绩大于 90 分的学生信息

Sno	Cno	Grade
201109001	001	93
201109001	002	83
201109001	003	52
201109002	001	NULL
201109002	002	70
201109002	003	NULL
201109003	001	74
201109003	002	69
201109004	001	NULL
201109004	002	89
201109004	003	85
201109005	001	79
201109005	002	90
201109005	003	45

SC表

Sno	Sname	Sex	Sage
201109001	李羽凡	男	20
201109002	王郁郁	女	21
201109003	聂乐乐	女	22
201109004	张东健	男	23
201109005	王子	男	24
201109006	邢星	女	25

Student表

Cno	Cname	Credit
001	数据结构	5
002	计算机网络	3
003	C#程序设计	5
004	数学	4
005	SQL Server 2008	4

Course表

图 10.2　SC 表、Student 表和 Course 表中的信息

10.4　带 IN 的嵌套查询

带 IN 的嵌套查询语法格式如下：

```
WHERE 查询表达式 IN(子查询)
```

一些嵌套内层的子查询会产生一个值，也有一些子查询会返回一系列值，即子查询不能返回带几行和几列数据的表。原因在于子查询的结果必须适合外层查询的语句。当子查询产生一系列值时，适合用带 IN 的嵌套查询。

把查询表达式单个数据和由子查询产生的一系列的数值相比较，如果数值匹配一系列值中的一个，则返回 TRUE。

【例 10.02】 在 Student 表和 SC 表中，查询参加考试的学生信息，SQL 语句及运行结果如图 10.3 所示。（**实例位置：资源包\源码\10\10.02**）

```
2 SELECT * FROM Student
3 WHERE Sno IN (SELECT Sno FROM SC )
```

结果 消息

	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	21
2	201109002	王都都	女	21
3	201109003	聂乐乐	女	22
4	201109004	张东健	男	22
5	201109005	王子	男	20

图 10.3 参加考试的学生的信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sno IN (SELECT Sno FROM SC )
```

10.5 带 NOT IN 的嵌套查询

NOT IN 的嵌套查询语法格式如下：

```
WHERE 查询表达式 NOT IN(子查询)
```

【例 10.03】 在 Course 表和 SC 表中，查询没有考试的课程信息，SQL 语句及运行结果如图 10.4 所示。（**实例位置：资源包\源码\10\10.03**）

SQL 语句如下：

```
SELECT * FROM Course
WHERE Cno NOT IN
(SELECT CNO FROM SC WHERE Cno IS NOT NULL)
```

查询过程是用主查询中 Cno 的值与子查询结果中的值比较，不匹配返回真值。由于主查询中的 004 和 005 的课程代号值与子查询的结果的数据不匹配，返回真值。所以查询结果显示 Cno 为 004 和 005 的课程信息。

【例 10.04】 在 Student 表和 SC 表中，查询没有考试的学生信息，SQL 语句及运行结果如图 10.5 所示。（**实例位置：资源包\源码\10\10.04**）

```
2 SELECT * FROM Course
3 WHERE Cno NOT IN
4 (SELECT CNO FROM SC WHERE Cno IS NOT NULL )
```

结果 消息

	Cno	Cname	Credit
1	004	数学	4
2	005	SQL Server 2008	4

图 10.4 没考试的课程信息

```
2 SELECT * FROM Student
3 WHERE Sno NOT IN
4 (SELECT Sno FROM SC WHERE Sno IS NOT NULL)
```

结果 消息

	Sno	Sname	Sex	Sage
1	201109006	邢星	女	20

图 10.5 没有参加考试的学生

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sno NOT IN
(SELECT Sno FROM SC WHERE Sno IS NOT NULL)
```

10.6　带 SOME 的嵌套查询

SQL 支持 3 种定量比较谓词：SOME、ANY 和 ALL。它们都是判断是否任何或全部返回值都满足搜索要求的。其中 SOME 和 ANY 谓词是存在量的，只注重是否有返回值满足搜索要求。这两种谓词含义相同，可以替换使用。

【例 10.05】　在 Student 表中，查询 Sage 小于平均年龄的所有学生的信息，SQL 语句及运行结果如图 10.6 所示。（**实例位置：资源包\源码\10\10.05**）

```
2 SELECT * FROM Student
3 WHERE Sage < SOME
4 (SELECT AVG(Sage) FROM Student)
```

结果　消息

	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	20
2	201109002	王都都	女	21

图 10.6　查询年龄小于平均年龄的学生信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sage < SOME
(SELECT AVG(Sage) FROM Student)
```

10.7　带 ANY 的嵌套查询

ANY 属于 SQL 支持的 3 种定量谓词之一，且和 SOME 完全等价，即能用 SOME 的地方完全可以使用 ANY。

【例 10.06】　在 Student 表中，查询 Sage 大于平均年龄的所有学生的信息，SQL 语句及运行结果如图 10.7 所示。（**实例位置：资源包\源码\10\10.06**）

```
2 SELECT * FROM Student
3 WHERE Sage > ANY
4 (SELECT AVG(Sage) FROM Student)
```

结果　消息

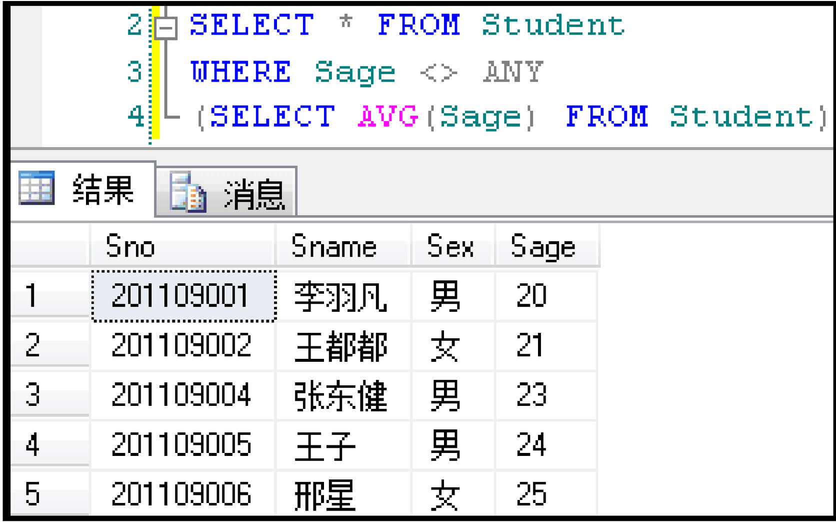
	Sno	Sname	Sex	Sage
1	201109004	张东健	男	23
2	201109005	王子	男	24
3	201109006	邢星	女	25

图 10.7　查询年龄大于平均年龄的学生信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sage > ANY
(SELECT AVG(Sage) FROM Student)
```

【例 10.07】 在 Student 表中，查询 Sage 不等于平均年龄的所有学生的信息，SQL 语句及运行结果如图 10.8 所示。（**实例位置：资源包\源码\10\10.07**）



	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	20
2	201109002	王都都	女	21
3	201109004	张东健	男	23
4	201109005	王子	男	24
5	201109006	邢星	女	25

图 10.8 查询年龄不等于平均年龄的学生信息

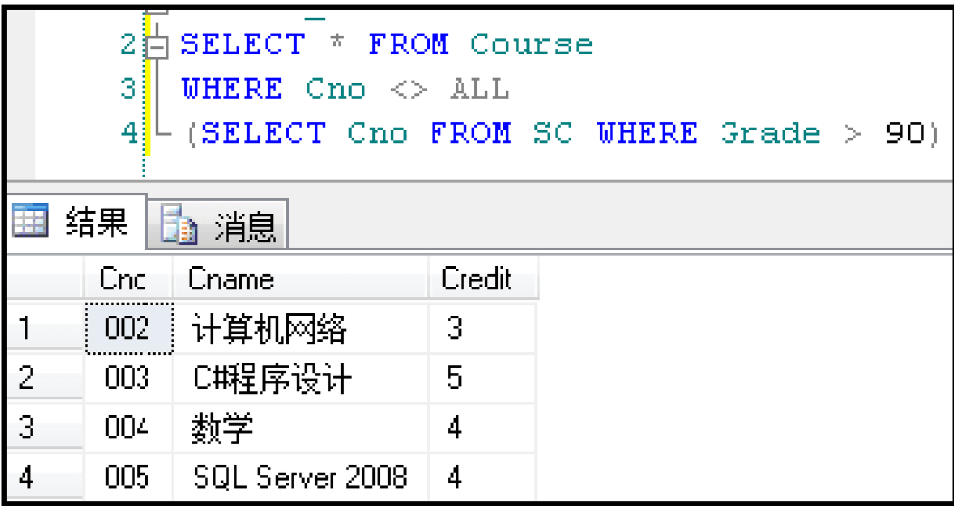
SQL 语句如下：

```
SELECT * FROM Student
WHERE Sage <> ANY
(SELECT AVG(Sage) FROM Student)
```

10.8 带 ALL 的嵌套查询

ALL 谓词的使用方法和 ANY 或者 SOME 谓词一样，也是把列值与子查询结果进行比较，但是它不要求任意结果值的列值为真，而是要求所有列的查询结果都为真，否则就不返回行。

【例 10.08】 在 SC 表中，查询 Grade 没有大于 90 分的 Cno 的详细信息，SQL 语句及运行结果如图 10.9 所示。（**实例位置：资源包\源码\10\10.08**）



	Cno	Cname	Credit
1	002	计算机网络	3
2	003	C#程序设计	5
3	004	数学	4
4	005	SQL Server 2008	4

图 10.9 查询某课程成绩没有大于 90 分的课程信息

SQL 语句如下：

```
SELECT * FROM Course
WHERE Cno <> ALL
(SELECT Cno FROM SC WHERE Grade > 90)
```

10.9　带 EXISTS 的嵌套查询

EXISTS 谓词只注重子查询是否返回行。如果子查询返回一个或多个行，谓词返回为真值，否则为假。EXISTS 搜索条件并不真正地使用子查询的结果。它仅仅测试子查询是否产生任何结果。

用带 IN 的嵌套查询也可以用带 EXISTS 的嵌套查询改写。

【例 10.09】　在 Student 表中，查询参加考试的学生信息，SQL 语句及运行结果如图 10.10 所示。（**实例位置：资源包\源码\10\10.09**）

```
2 SELECT * FROM Student
3 WHERE EXISTS
4 (SELECT Sno FROM SC WHERE Student.Sno = SC.Sno)
```

结果　消息

	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	20
2	201109002	王都都	女	21
3	201109003	聂乐乐	女	22
4	201109004	张东健	男	23
5	201109005	王子	男	24

图 10.10　参加考试的学生信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE EXISTS
(SELECT Sno FROM SC WHERE Student.Sno = SC.Sno)
```

【例 10.10】　在 Student 表中，查询没有参加考试的学生信息，SQL 语句及运行结果如图 10.11 所示。（**实例位置：资源包\源码\10\10.10**）

```
2 SELECT * FROM Student
3 WHERE NOT EXISTS
4 (SELECT Sno FROM SC WHERE Student.Sno = SC.Sno)
```

结果　消息

	Sno	Sname	Sex	Sage
1	201109006	邢星	女	25

图 10.11　没有参加考试的学生信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE NOT EXISTS
(SELECT Sno FROM SC WHERE Student.Sno = SC.Sno)
```

10.10　小　　结

本章主要对 SQL 中的高级数据查询进行了详细讲解，在具体讲解过程中，由于嵌套查询中必然用到子查询，因此首先介绍了子查询和嵌套查询的概念，然后对各种嵌套查询进行了详细讲解。学习本章内容时，应该重点掌握各种嵌套查询的使用。

第2篇

提高篇

本篇介绍了索引与数据完整性、流程控制、存储过程、触发器、游标的使用、SQL 中的事务、SQL Server 高级开发、SQL Server 安全管理、SQL Server 维护管理等内容。学习完本篇，读者将能够掌握比较高级的 SQL 及 SQL Server 管理知识，并对数据库进行管理。

第11章

索引与数据完整性

（视频讲解：56分钟）

本章主要介绍索引与数据完整性，主要包括索引的概念、索引的创建、索引的删除、索引的分析与维护、域完整性、实体完整性和引用完整性。通过本章的学习，读者应掌握建立或者删除索引的方法，能够使用索引优化数据库查询，熟悉数据完整性。

学习摘要：

- 索引的基本概念及分类
- 索引的优缺点
- 创建、修改和删除索引
- 索引的分析与维护
- 使用数据库进行全文索引
- 全文目录的创建与删除
- 数据完整性

11.1　索引的概念



与书中的索引一样，数据库中的索引使用户可以快速找到表或索引视图中的特定信息。索引包含从表或视图中一个或多个列生成的键，以及映射到指定数据的存储位置的指针。通过创建设计良好的索引以支持查询，可以显著提高数据库查询和应用程序的性能。索引可以减少为返回查询结果集而必须读取的数据量。索引还可以强制表中的行具有唯一性，从而确保表数据的数据完整性。

索引是一个单独的、物理的数据库结构，在 SQL Server 中，索引是为了加速对表中数据行的检索而创建的一种分散存储结构。它是针对一个表而建立的，每个索引页面中的行都含有逻辑指针，指向数据表中的物理位置，以便加速检索物理数据。因此，对表中的列是否创建索引，将对查询速度有很大的影响。一个表的存储是由两部分组成的，一部分用来存放表的数据页，另一部分存放索引页。通常索引页面对于数据页来说小得多。在进行数据检索时，系统首先搜索索引页面，从中找到所需数据的指针，然后直接通过该指针从数据页面中读取数据，从而提高查询速度。

11.2　索引的优缺点



索引是与表或视图关联的磁盘上结构，可以加快从表或视图中检索行的速度。本节将介绍索引的优缺点。

11.2.1　索引的优点

索引有以下优点。

- ☑ 创建唯一性索引，保证数据库表中每一行数据的唯一性。
- ☑ 大大加快数据的检索速度，这也是创建索引的最主要原因。
- ☑ 加速表与表之间的连接，特别是在实现数据的参考完整性方面特别有意义。
- ☑ 在使用分组和排序子句进行数据检索时，同样可以减少查询中分组和排序的时间。
- ☑ 通过使用索引，可以在查询的过程中使用优化隐藏器，提高系统的性能。

11.2.2　索引的缺点

索引有以下缺点。

- ☑ 创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。
- ☑ 索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚集索引，那么需要的空间就会更大。
- ☑ 当对表中的数据进行增加、删除和修改的时候，索引也要动态地维护，降低了数据的维护速度。



11.3　索引的分类

在 SQL Server 2014 中提供的索引类型主要有聚集索引、非聚集索引、唯一索引、包含性列索引、索引视图、全文索引、空间索引、筛选索引和 XML 索引。

按照存储结构的不同，可以将索引分为聚集索引和非聚集索引两类。

11.3.1　聚集索引

聚集索引根据数据行的键值在表或视图中排序和存储这些数据行。索引定义中包含聚集索引列。每个表只能有一个聚集索引，因为数据行本身只能按一个顺序排序。

只有当表包含聚集索引时，表中的数据行才按排序顺序存储。如果表具有聚集索引，则该表称为聚集表。如果表没有聚集索引，则其数据行存储在一个称为堆的无序结构中。

除了个别表之外，每个表都应该有聚集索引。聚集索引除了可以提高查询性能之外，还可以按需重新生成或重新组织来控制表碎片。

聚集索引按下列方式实现。

（1）PRIMARY KEY 和 UNIQUE 约束

☑ 在创建 PRIMARY KEY 约束时，如果不存在该表的聚集索引且未指定唯一非聚集索引，则将自动对一列或多列创建唯一聚集索引。主键列不允许空值。

☑ 在创建 UNIQUE 约束时，默认情况下将创建唯一非聚集索引，以便强制 UNIQUE 约束。如果不存在该表的聚集索引，则可以指定唯一聚集索引。

（2）独立于约束的索引

指定非聚集主键约束后，用户可以对非主键列的列创建聚集索引。

（3）索引视图

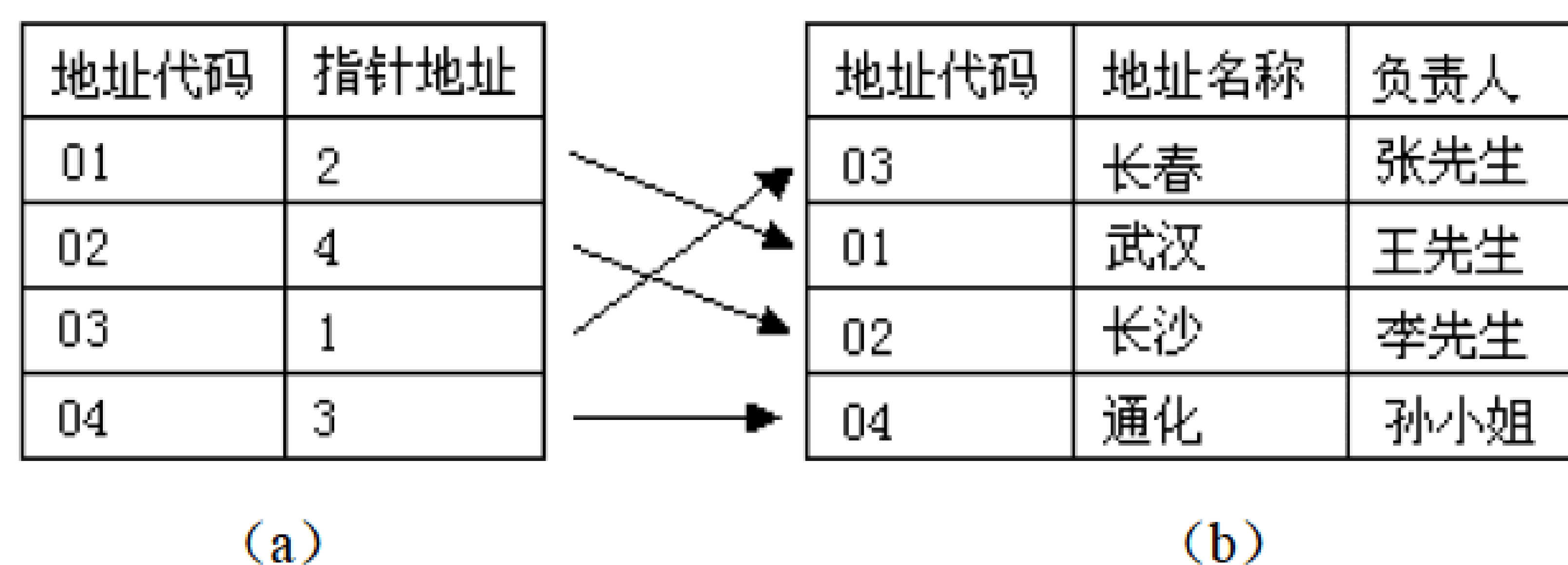
若要创建索引视图，请对一个或多个视图列定义唯一聚集索引。视图将具体化，并且结果集存储在该索引的页级别中，其存储方式与表数据存储在聚集索引中的方式相同。

11.3.2　非聚集索引

非聚集索引具有独立于数据行的结构。非聚集索引包含非聚集索引键值，并且每个键值项都有指向包含该键值的数据行的指针。

从非聚集索引中的索引行指向数据行的指针称为行定位器。行定位器的结构取决于数据页是存储在堆中还是聚集表中。对于堆，行定位器是指向行的指针。对于聚集表，行定位器是聚集索引键。

下面以图 11.1 对非聚集索引的结构进行详细的说明。图 11.1（a）中的数据是按图 11.1（b）中的数据进行顺序存储的，在图 11.1（a）中为“地址代码”列建立索引，“指针地址”列是每条记录在表中的存储位置（通常称为指针），当查询地址代码为 01 的信息时，先在索引表中查找地址代码 01，然后根据索引表中的指针地址（在这里指针地址为 2）找到第 2 条记录，这样就很大地提高了查询速度。



地址代码	指针地址
01	2
02	4
03	1
04	3

（a）

地址代码	地址名称	负责人
03	长春	张先生
01	武汉	王先生
02	长沙	李先生
04	通化	孙小姐

（b）

图 11.1　非聚集索引结构图



11.4　索引的操作

索引就是加快检索表中数据的方法。它对数据表中一个或多个列的值进行结构排序，是数据库中一个非常有用的对象。本节主要介绍如何通过 SQL Server Management Studio 和 Transact-SQL 语句创建索引。

11.4.1　索引的创建

1．使用 SQL Server Management Studio 创建索引

操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 数据库。

（2）选择指定的数据库 db_2014，然后展开要创建索引的表，在表的下级菜单中，鼠标右键单击“索引”，在弹出的快捷菜单中选择“新建索引”命令，如图 11.2 所示。弹出“新建索引”窗口，如图 11.3 所示。



图 11.2　选择“新建索引”命令

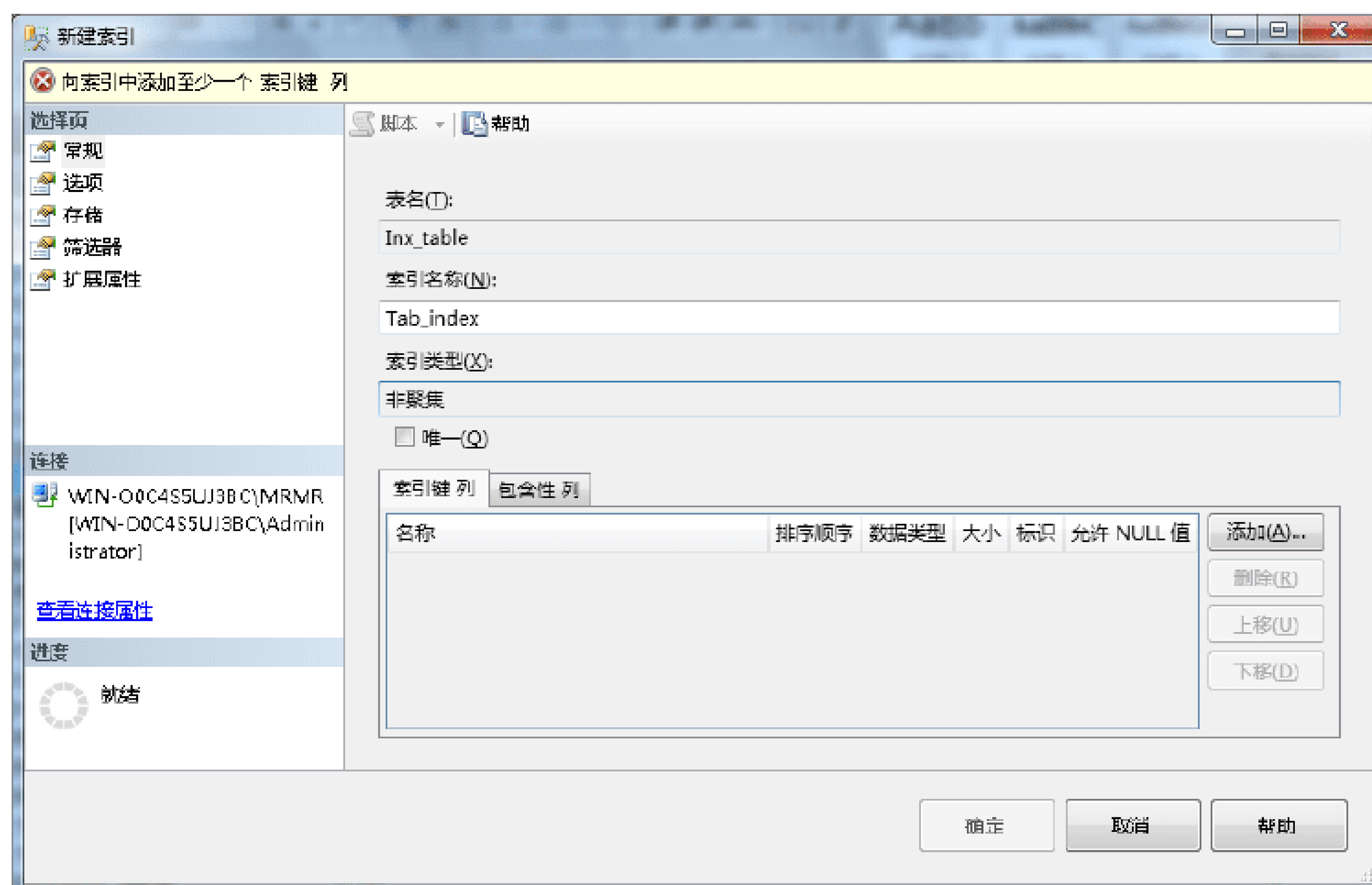


图 11.3　“新建索引”窗口

（3）在“新建索引”窗口中单击“添加”按钮，弹出“从表中选择列”窗口，在该窗口中选择要添加到索引键的表列，如图 11.4 所示。

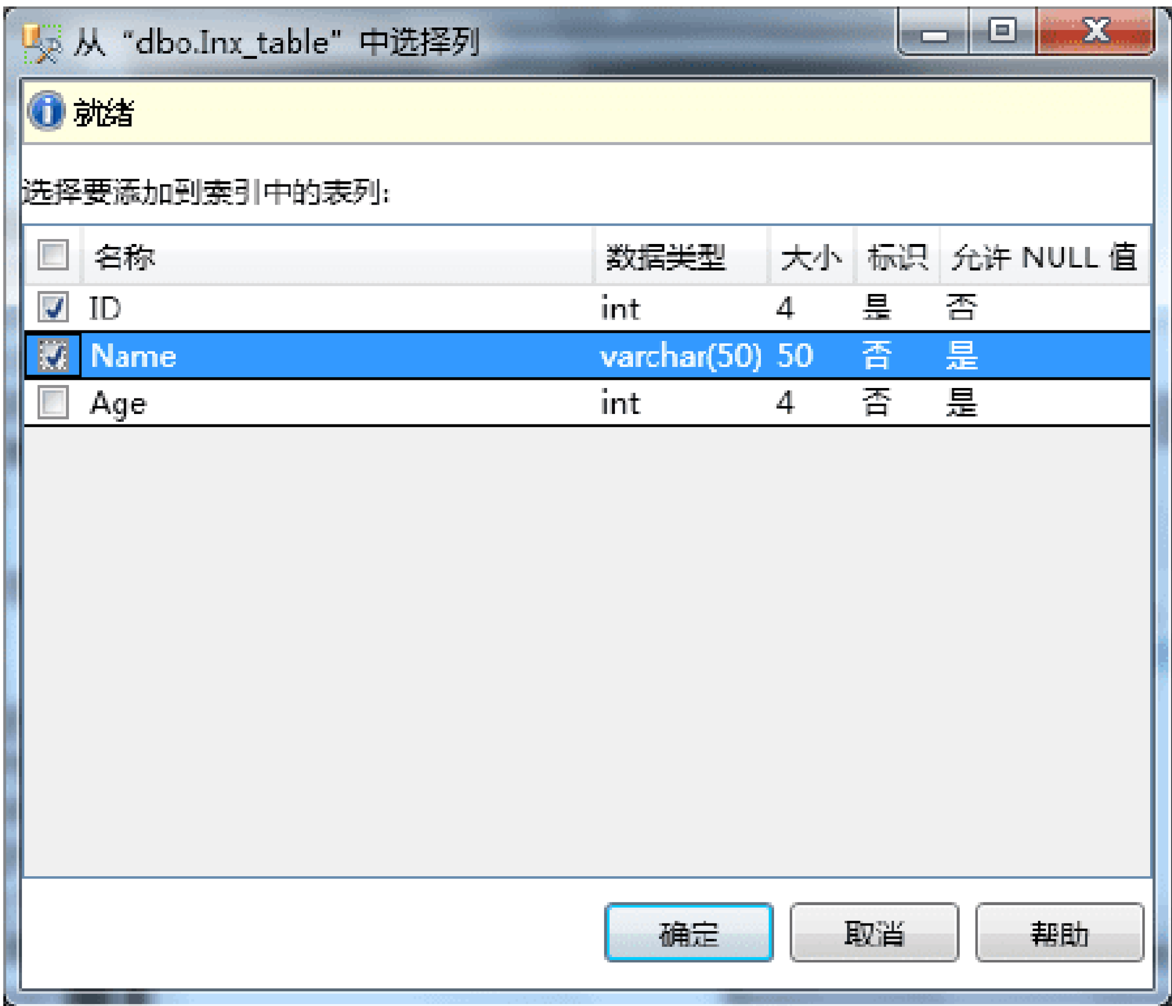


图 11.4 “从表中选择列”窗口

（4）单击“确定”按钮，返回到“新建索引”窗口，单击“确定”按钮，便完成了索引的创建。

2．使用 Transact-SQL 语句创建索引

CREATE INDEX 语句为给定表或视图创建一个改变物理顺序的聚集索引，也可以创建一个具有查询功能的非聚集索引。

语法格式如下：

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX index_name
    ON {table | view} (column [ASC | DESC] [,...n])
[WITH < index_option > [,...n]]
[ON filegroup]
< index_option > ::=
  {PAD_INDEX |
     FILLFACTOR = fillfactor |
     IGNORE_DUP_KEY |
     DROP_EXISTING |
  STATISTICS_NORECOMPUTE |
  SORT_IN_TEMPDB
}
```

CREATE INDEX 语句的参数及说明如表 11.1 所示。

表 11.1　CREATE INDEX 语句的参数及说明

参　　数	描　　述
[UNIQUE][CLUSTERED | NONCLUSTERED]	指定创建索引的类型，参数依次为唯一索引、聚集索引和非聚集索引。当省略 UNIQUE 选项时，建立非唯一索引，省略 CLUSTERED|NONCLUSTERED 选项时，建立聚集索引，省略 NONCLUSTERED 选项时，建立唯一聚集索引
index_name	索引名。索引名在表或视图中必须唯一，但在数据库中不必唯一。索引名必须遵循标识符规则

续表

参　　数	描　　述
table	包含要创建索引的列的表。可以选择指定数据库和表所有者
column	应用索引的列。指定两个或多个列名，可为指定列的组合值创建组合索引
[ASC | DESC]	确定具体某个索引列的升序或降序排序方向。默认设置为 ASC
PAD_INDEX	指定索引中间级中每个页（节点）上保持开放的空间
FILLFACTOR	指定在 SQL Server 创建索引的过程中，各索引页的填满程度
IGNORE_DUP_KEY	控制向唯一聚集索引的列插入重复的键值时所发生的情况。如果为索引指定了 IGNORE_DUP_KEY，并且执行了创建重复键的 INSERT 语句，SQL Server 将发出警告消息并忽略重复的行
DROP_EXISTING	指定应删除并重建已命名的先前存在的聚集索引或非聚集索引
SORT_IN_TEMPDB	指定用于生成索引的中间排序结果将存储在 tempdb 数据库中
ON filegroup	在给定的文件组上创建指定的索引。该文件组必须已创建

【例 11.01】　为 Student 表的 Sno 列创建非聚集索引。（**实例位置：资源包\源码\ 11\11.01**）

SQL 语句如下：

```
USE db_2014
CREATE INDEX IX_Stu_Sno
ON Student (Sno)
```

【例 11.02】　为 Student 表的 Sno 列创建唯一聚集索引。（**实例位置：资源包\源码\11\11.02**）

SQL 语句如下：

```
USE db_2014
CREATE UNIQUE CLUSTERED INDEX IX_Stu_Sno1
ON Student (Sno)
```



注意

无法对表创建多个聚集索引。

【例 11.03】　为 Student 表的 Sno 列创建组合索引。（**实例位置：资源包\源码\ 11\11.03**）

SQL 语句如下：

```
USE db_2014
CREATE INDEX IX_Stu_Sno2
ON Student (Sno,Sname DESC)
```

【例 11.04】　用 FILLFACTOR 参数为 Student 表的 Sno 列创建一个填充因子为 100 的非聚集索引。（**实例位置：资源包\源码\11\11.04**）

SQL 语句如下：

```
USE db_2014
CREATE NONCLUSTERED INDEX IX_Stu_Sno3
ON Student (Sno)
WITH FILLFACTOR = 100
```

【例 11.05】 用 IGNORE_DUP_KEY 参数为 Student 表的 Sno 列创建唯一聚集索引，并且不能输入重复值。（**实例位置：资源包\源码\11\11.05**）

SQL 语句如下：

```
USE db_2014
CREATE UNIQUE CLUSTERED INDEX IX_Stu_Sno4
ON Student (Sno)
WITH IGNORE_DUP_KEY
```

3. 创建索引的原则

使用索引虽然可以提高系统的性能，增强数据的检索速度，但它需要占用大量的物理存储空间，建立索引的一般原则如下。

（1）只有表的所有者可以在同一个表中创建索引。

（2）每个表中只能创建一个聚集索引。

（3）每个表中最多可以创建 249 个非聚集索引。

（4）在经常查询的字段上建立索引。

（5）定义 text、image 和 bit 数据类型的列上不要建立索引。

（6）在外键列上可以建立索引。

（7）主键列上一定要建立索引。

（8）在那些重复值比较多、查询较少的列上不要建立索引。

11.4.2 查看索引信息

1. 使用 SQL Server Management Studio 器查看索引

使用 SQL Server Management Studio 查看索引的步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 数据库。

（2）选择指定的数据库 db_2014，然后展开要查看索引的表。

（3）鼠标右键单击该表，在弹出的快捷菜单中选择“设计”命令。

（4）弹出表设计窗口，鼠标右键单击该窗口，在弹出的快捷菜单中选择“索引/键”命令。

（5）打开“索引/键”窗体，如图 11.5 所示。在窗体的左侧选中某个索引，在窗体的右侧就可以查看此索引的信息，并可以修改相关的信息。

2. 使用系统存储过程查看索引

系统存储过程 sp_helpindex 可以报告有关表或视图上索引的信息。

语法格式如下：

```
sp_helpindex [@objname =] 'name'
```

参数[@objname =] 'name'表示用户定义的表或视图的限定或非限定名称。

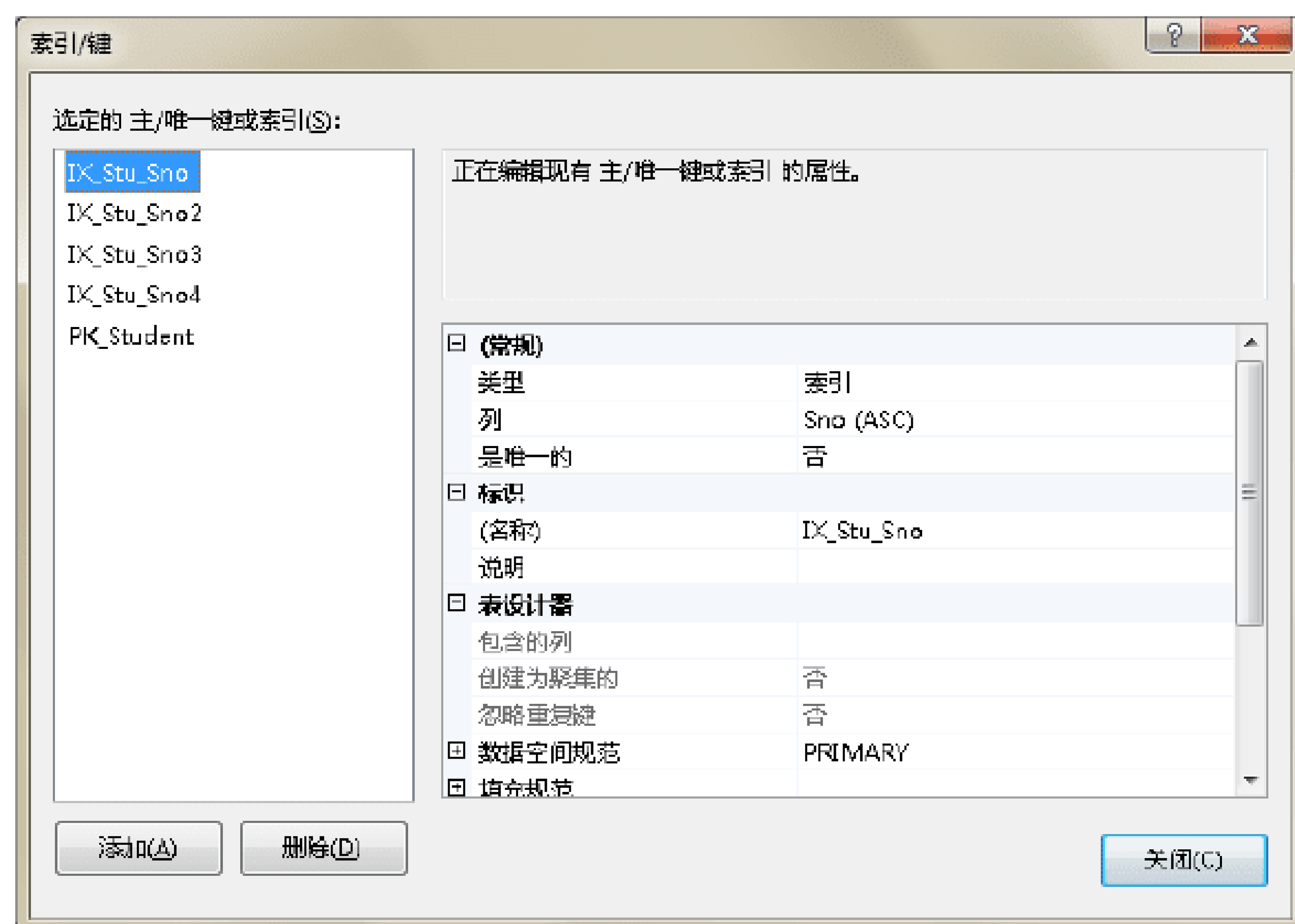


图 11.5　“索引/键”窗体

【例 11.06】　用系统存储过程 Sp_helpindex，查看 db_2014 数据库中 Student 表的索引信息。（**实例位置：资源包\源码\11\11.06**）

SQL 语句如下：

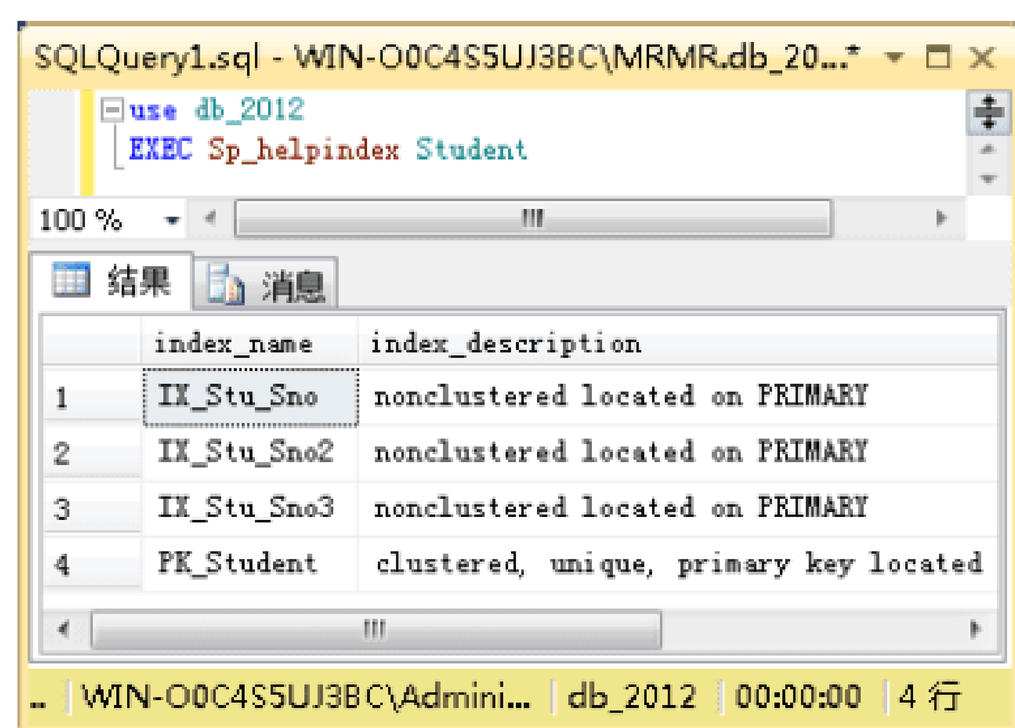
```
use db_2014
EXEC Sp_helpindex Student
```

运行结果如图 11.6 所示。

3．利用系统表查看索引信息

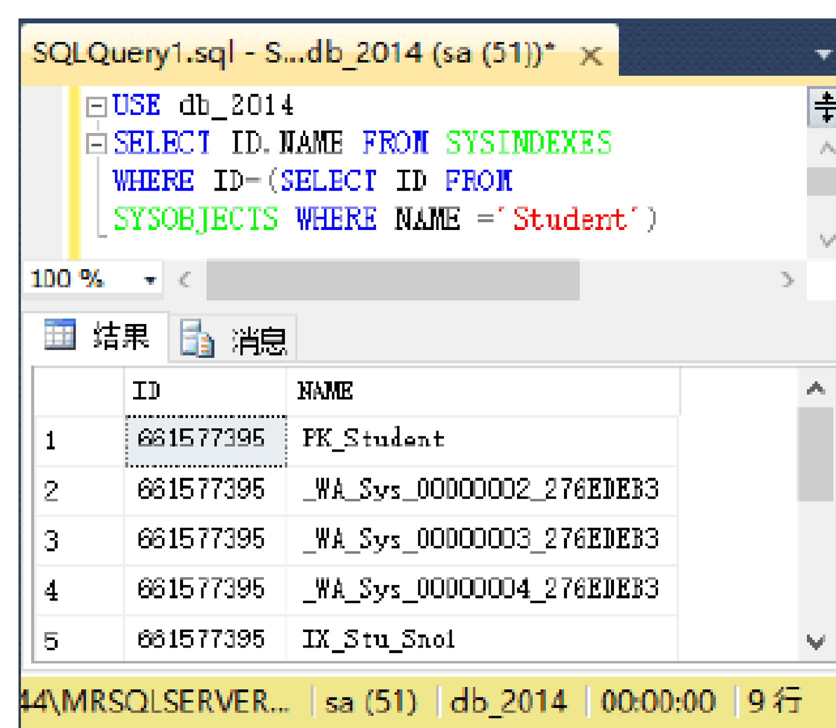
查看数据库中指定表的索引信息，可以利用该数据库中的系统表 sysobjects（记录当前数据库中所有对象的相关信息）和 sysindexes（记录有关索引和建立索引表的相关信息）进行查询，系统表 sysobjects 可以根据表名查找到索引表的 ID 号，再利用系统表 sysindexes 根据 ID 号查找到索引文件的相关信息。

【例 11.07】　利用系统表查看 db_2014 数据库中 Student 表中的索引信息，SQL 语句及运行结果如图 11.7 所示。（**实例位置：资源包\源码\11\11.07**）



	index_name	index_description
1	IX_Stu_Sno	nonclustered located on PRIMARY
2	IX_Stu_Sno2	nonclustered located on PRIMARY
3	IX_Stu_Sno3	nonclustered located on PRIMARY
4	PK_Student	clustered, unique, primary key located

图 11.6　使用系统存储过程查看索引



	ID	NAME
1	661577395	PK_Student
2	661577395	_WA_Sys_00000002_276EDEB3
3	661577395	_WA_Sys_00000003_276EDEB3
4	661577395	_WA_Sys_00000004_276EDEB3
5	661577395	IX_Stu_Sno1

图 11.7　查看 Student 表中的索引

SQL 语句如下：

```
USE db_2014
SELECT ID,NAME FROM SYSINDEXES
```

```
WHERE ID=(SELECT ID FROM
SYSOBJECTS WHERE NAME ='Student')
```

11.4.3 索引的修改

1．使用 SQL Server Management Studio 修改索引

使用 SQL Server Management Studio 修改索引与使用它查看索引的步骤相同，在“索引/键”窗体中就可以修改索引的相关信息。

2．使用 Transact-SQL 语句更改索引名称

在当前数据库中更改用户创建对象的名称。此对象可以是表、索引、列、别名数据类型或 Microsoft .NET Framework 公共语言运行时（CLR）用户定义类型。

语法格式如下：

```
sp_rename [@objname =] 'object_name',
[@newname =] 'new_name'
[, [@objtype =] 'object_type']
```

参数说明如下。

☑ [@objname =] 'object_name'：用户对象或数据类型的当前限定或非限定名称。

☑ [@newname =] 'new_name'：指定对象的新名称。

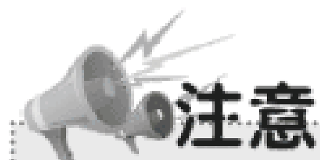
☑ [@objtype =] 'object_type'：要重命名的对象的类型。

【例 11.08】 利用系统存储过程 sp_rename，IX_Stu_Sno 索引重命名为 IX_Stu_Sno1。（**实例位置：资源包\源码\11\11.08**）

SQL 语句如下：

```
USE db_2014
EXEC sp_rename 'Student.IX_Stu_Sno','IX_Stu_Sno1'
```

运行结果如图 11.8 所示。



注意

要对索引进行重命名时，需要修改的索引名格式必须为“表名.索引名”。

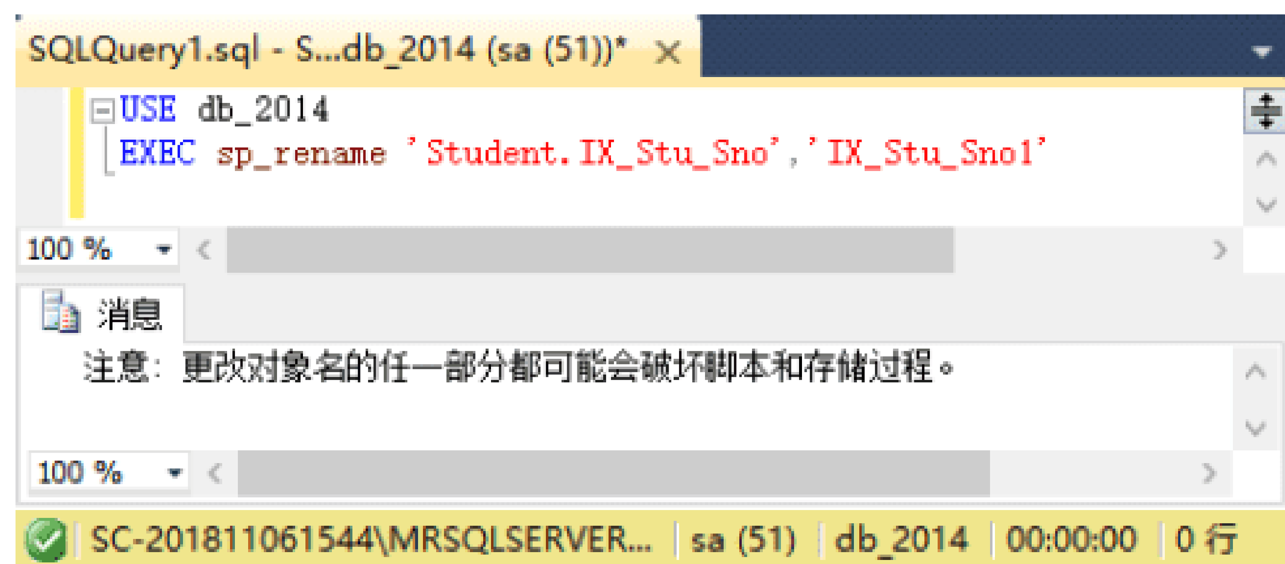


图 11.8 更改索引名称

11.4.4 索引的删除

1．使用 SQL Server Management Studio 器删除索引

使用 SQL Server Management Studio 删除索引与使用它查看索引的步骤相同，在“索引/键”窗体

中，单击“删除”按钮，就可以把当前选中的索引删除。

2. 使用 Transact-SQL 语句删除索引

DROP INDEX 语句表示从当前数据库中删除一个或多个关系索引、空间索引、筛选索引或 XML 索引。

DROP INDEX 语句不适用于通过定义 PRIMARY KEY 或 UNIQUE 约束创建的索引。若要删除该约束和相应的索引，请使用带有 DROP CONSTRAINT 子句的 ALTER TABLE。

DROP INDEX 语句的语法格式如下：

```
DROP INDEX
{<drop_relational_or_xml_or_spatial_index> [,...n]
| <drop_backward_compatible_index> [,...n]
}
<drop_relational_or_xml_or_spatial_index> ::=
    index_name ON <object>
    [WITH (<drop_clustered_index_option> [,...n])]
<drop_backward_compatible_index> ::=
    [owner_name.] table_or_view_name.index_name
<object> ::=
{
    [database_name. [schema_name] . | schema_name.]
    table_or_view_name
}
```

DROP INDEX 语句的参数及说明如表 11.2 所示。

表 11.2　DROP INDEX 语句的参数及说明

参　　数	描　　述
index_name	要删除的索引名称
database_name	数据库的名称
schema_name	该表或视图所属架构的名称
table_or_view_name	与该索引关联的表或视图的名称
<drop_clustered_index_option>	控制聚集索引选项。这些选项不能与其他索引类型一起使用

【例 11.09】　删除 Student 表中的 IX_Stu_Sno1 索引。（**实例位置：资源包\源码\11\11.09**）

SQL 语句如下：

```
USE db_2014
--判断表中是否有要删除的索引
If EXISTS(Select * from sysindexes where name='IX_Stu_Sno1')
 Drop Index Student.IX_Stu_Sno1
```

运行结果如图 11.9 所示。

【例 11.10】　删除 Student 表中的 IX_Stu_Sno3 索引和 SC 表中的 IX_SC_Sno 索引。（**实例位置：资源包\源码\11\11.10**）

SQL 语句如下：

```
USE db_2014
Drop Index Student.IX_Stu_Sno3,SC.IX_SC_Sno
```

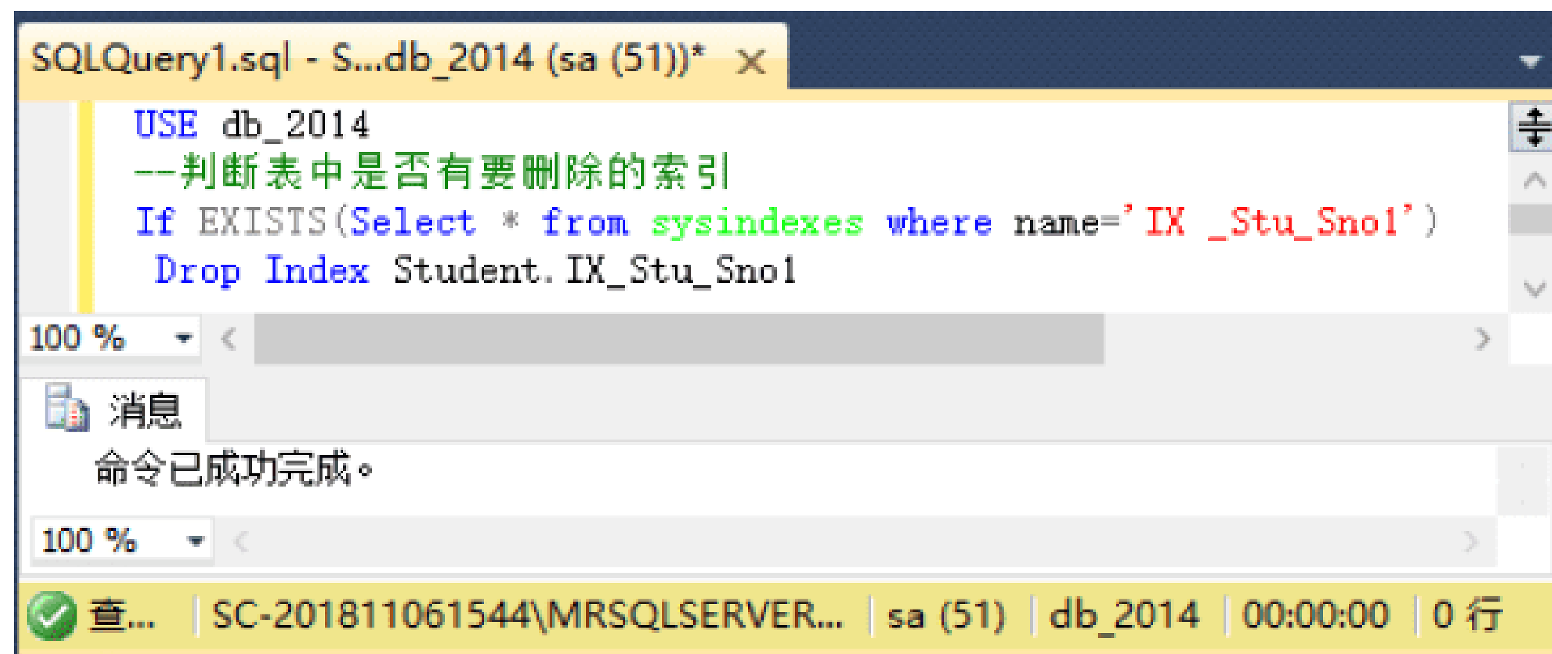


图 11.9　索引的删除

11.4.5　设置索引的选项

1．设置 PAD_INDEX 选项

PAD_INDEX 选项是设置创建索引期间中间级别页中可用空间的百分比。

对于非叶级索引页需要使用 PAD_INDEX 选项设置其预留空间的大小。PAD_INDEX 选项只有在指定了 FILLFACTOR 选项时才有用，因为 PAD_INDEX 是由 FILLFACTOR 所指定的百分比决定。默认情况下，给定中间级页上的键集，SQL Server 将确保每个索引页上的可用空间至少可以容纳一个索引允许的最大行。如果 FILLFACTOR 指定的百分比不够大，无法容纳一行，SQL Server 将在内部使用允许的最小值替代该百分比。

【例 11.11】 为 Student 表的 Sno 列创建一个簇索引 IX_Stu_Sno，并将预留空间设置为 10。（**实例位置：资源包\源码\11\11.11**）

SQL 语句如下：

```
USE db_2014
CREATE UNIQUE CLUSTERED INDEX IX_Stu_Sno
ON Student(Sno)
WITH PAD_INDEX,FILLFACTOR = 10
```

2．设置 FILLFACTOR 选项

FILLFACTOR 选项是设置创建索引期间每个索引页的页级别中可用空间的百分比。

数据库系统在存储数据库文件时，有时会将用到的数据页隔断，在使用数据索引的同时会产生一定程度的碎片。为了尽量减少页拆分，在创建索引时，可以选择 FILLFACTOR（称为填充因子）选项，此选项用来指定各索引页的填满程度，即指定索引页上所留出的额外的间隙和保留一定的百分比空间，从而扩充数据的存储容量和减少页拆分。FILLFACTOR 选项的取值范围是 1～100，表示用户创建索引时数据容量所占页容量的百分比。

【例 11.12】 在 db_2014 数据库中的 Student 表上创建基于 Sname 列的非聚集索引 IX_Stu_Snanme，并且为升序，填充因子为 80。（**实例位置：资源包\源码\11\11.12**）

SQL 语句如下：

```
USE db_2014
GO
CREATE INDEX IX_Stu_Sname ON Student(Sname)
WITH FILLFACTOR=80
GO
```

3．设置 ASC/DESC 选项

排序查询是指将查询结果按指定属性的升序（ASC）或降序（DESC）排列，由 ORDER BY 子句指明。ASC/DESC 选项可以在创建索引时设置索引方式。

【例 11.13】　在 Student 表中创建一个聚集索引 MR_Stu_Sage，将 Sage 列按从小到大排序。(**实例位置：资源包\源码\11\11.13**)

SQL 语句如下：

```
USE db_2014
CREATE CLUSTERED INDEX MR_Stu_Sage
ON Student (Sage DESC)
```

创建索引后，数据表如图 11.10 所示。

Sno	Sname	Sex	Sage
201109001	李羽凡	男	19
201109002	王都都	女	22
20110[illegible]	[illegible]	[illegible]	23
20110[illegible]	[illegible]	[illegible]	22
201109005	王子	男	23
201109006	邢星	女	26
201109007	触发器	男	23

对 Sage 字段进行排序前

Sno	Sname	Sex	Sage
201109006	邢星	女	26
201109007	触发器	男	23
2011[illegible]	[illegible]	[illegible]	23
2011[illegible]	[illegible]	[illegible]	23
201109004	张东健	男	22
201109002	王都都	女	22
201109001	李羽凡	男	19

对 Sage 字段进行降序排序

图 11.10　对 Sage 字段进行排序

【例 11.14】　在 Student 表中创建一个聚集索引 MR_Stu，将 Sage 列按从大到小排序，Sno 列从小到大排序。(**实例位置：光盘\源码\11\11.14**)

SQL 语句如下：

```
USE db_2014
CREATE CLUSTERED INDEX MR_Stu
ON Student (Sage DESC,Sno ASC)
```

创建索引后，数据表如图 11.11 所示。

Sno	Sname	Sex	Sage
201109006	邢星	女	26
201109003	聂乐乐	女	23
201109005	王子	男	23
201109007	触发器	男	23
201109002	[illegible]	[illegible]	22
201109004	[illegible]	[illegible]	22
201109001	[illegible]	[illegible]	19

先按 Sage 的降序再按 Sno 的升序排序排序

图 11.11　按多字段进行排序

4．设置 SORT_IN_TEMPDB 选项

SORT_IN_TEMPDB 选项是确定对创建索引期间生成的中间排序结果进行排序的位置。如果为 ON，则排序结果存储在 tempdb 中。如果为 OFF，则排序结果存储在存储结果索引的文件组或分区方案中。

【例 11.15】 用 SORT_IN_TEMPDB 选项创建 MR_Stu 索引，当 tempdb 与用户数据库位于不同的磁盘集上时，可以减少创建索引所需的时间。（**实例位置：资源包\源码\11\11.15**）

SQL 语句如下：

```
CREATE UNIQUE CLUSTERED INDEX MR_Stu_Sno
ON Student (Sno ASC)
WITH SORT_IN_TEMPDB
```

5．设置 STATISTICS_NORECOMPUTE 选项

STATISTICS_NORECOMPUTE 选项指定是否应自动重新计算过期的索引统计信息。

【例 11.16】 在 Student 表上创建索引 MR_Stu，其功能是不自动重新计算过期的索引统计信息。（**实例位置：资源包\源码\11\11.16**）

SQL 语句如下：

```
USE db_2014
CREATE UNIQUE CLUSTERED INDEX MR_Stu
ON Student (Sno ASC)
WITH STATISTICS_NORECOMPUTE
```

6．设置 UNIQUE 选项

UNIQUE 选项是确定是否允许并发用户在索引操作期间访问基础表或聚集索引数据以及任何关联非聚集索引。

为表或视图创建唯一索引（不允许存在索引值相同的两行）。视图上的聚集索引必须是 UNIQUE 索引。如果存在唯一索引，当使用 UPDATE 或 INSERT 语句产生重复值时将回滚，并显示错误信息。即使 UPDATE 或 INSERT 语句更改了许多行但只产生了一个重复值，也会出现这种情况。如果在有唯一索引并且指定了 IGNORE_DUP_KEY 子句情况下输入数据，则只有违反 UNIQUE 索引的行才会失败。在处理 UPDATE 语句时，IGNORE_DUP_KEY 不起作用。

【例 11.17】 用 IGNORE_DUP_KEY 参数创建唯一聚集索引，并且不能输入重复值，改变行的物理排序。（**实例位置：资源包\源码\11\11.17**）

SQL 语句如下：

```
USE db_2014
CREATE UNIQUE CLUSTERED INDEX MR_Stu_Sno ON Student (Sno)
WITH IGNORE_DUP_KEY
```

7．设置 DROP_EXISTING 选项

DROP_EXISTING 选项指示应删除和重新创建现有索引。

删除 SQL Server 2014 中已存在的索引，并根据修改重新创建一个索引，如果创建的是一个聚集索

引，并且被索引的表上还存在其他非聚集索引，通过创建可以提高表的查询性能，因为重建聚集索引将强制重建所有的非聚集索引。

【例 11.18】 对已有的索引 MR_Stu，进行重新创建。(**实例位置：资源包\源码\11\11.18**)

SQL 语句如下：

```
CREATE UNIQUE CLUSTERED INDEX MR_Stu
ON Student (Sno ASC)
WITH DROP_EXISTING
```

11.5　索引的分析与维护


视频讲解

索引建立后，还需对它们进行分析和维护。本节主要讲解索引的分析及维护的方法。

11.5.1　索引的分析

1．使用 SHOWPLAN 语句

SHOWPLAN 语句用来显示查询语句的执行信息，包含查询过程中连接表时所采取的每个步骤以及选择哪个索引。语法格式如下：

```
SET SHOWPLAN_ALL {ON | OFF}
SET SHOWPLAN_TEXT {ON | OFF}
```

参数说明如下。

☑　ON：显示查询执行信息。

☑　OFF：不显示查询执行信息（系统默认）。

SET SHOWPLAN_ALL 的设置是在执行或运行时设置，而不是在分析时设置。如果 SET SHOWPLAN_ALL 为 ON，则 SQL Server 将返回每个语句的执行信息但不执行语句。Transact-SQL 语句不会被执行。在将此选项设置为 ON 后，将始终返回有关所有后续 Transact-SQL 语句的信息，直到将该选项设置为 OFF 为止。

SET SHOWPLAN_TEXT 的设置是在执行或运行时设置的，而不是在分析时设置的。当 SET SHOWPLAN_TEXT 为 ON 时，SQL Server 将返回每个 Transact-SQL 语句的执行信息，但不执行语句。将该选项设置为 ON 以后，将返回有关所有后续 SQL Server 语句的执行计划信息，直到将该选项设置为 OFF 为止。

【例 11.19】 在 db_2014 数据库的 Student 表中查询所有性别为男且年龄大于 23 岁的学生信息。(**实例位置：资源包\源码\11\11.19**)

SQL 语句如下：

```
USE db_2014
GO
SET SHOWPLAN_ALL ON
```

```
GO
SELECT Sname,Sex,Sage FROM Student WHERE Sex='男' AND Sage >23
GO
SET SHOWPLAN_ALLOFF
GO
```

运行结果如图 11.12 所示。

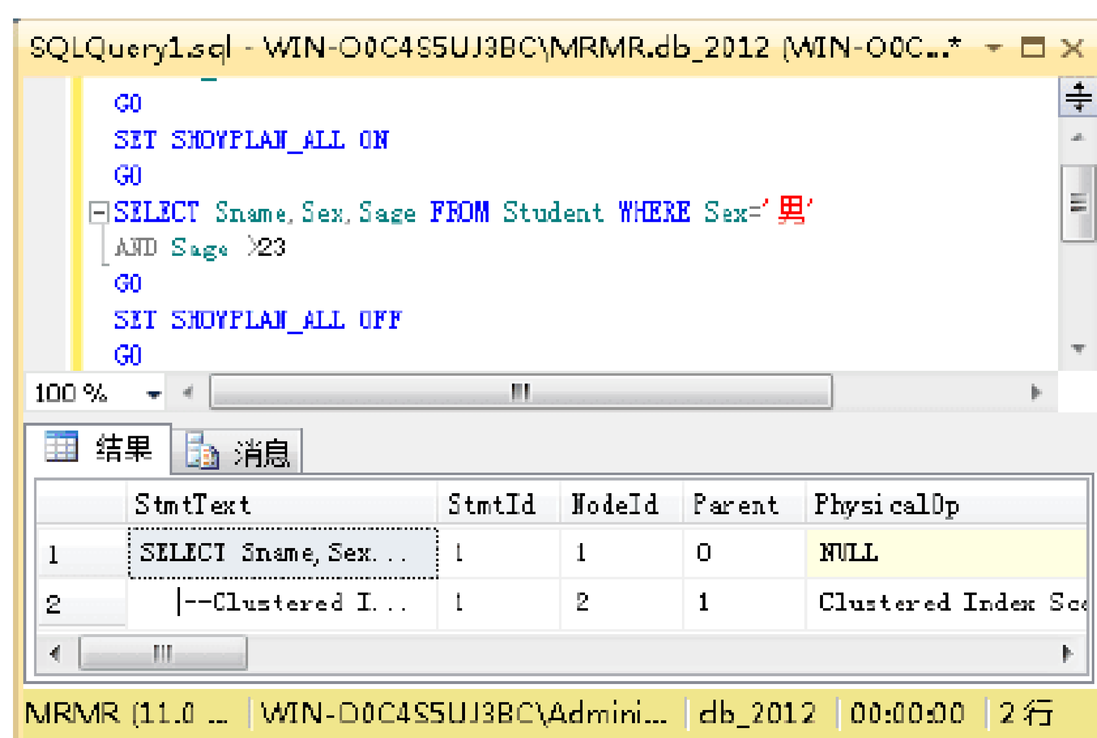


图 11.12　SHOWPLAN 语句的使用

2．使用 STATISTICS IO 语句

STATISTICS IO 语句表示使 SQL Server 显示有关由 Transact-SQL 语句生成的磁盘活动量的信息。语法格式如下：

```
SET STATISTICS IO {ON | OFF}
```

如果 STATISTICS IO 为 ON，则显示统计信息。如果为 OFF，则不显示统计信息。如果将此选项设置为 ON，则所有后续的 Transact-SQL 语句将返回统计信息，直到将该选项设置为 OFF 为止。

【例 11.20】 在 db_2014 数据库中的 Student 表中查询所有性别为男且年龄大于 20 岁的学生信息，并显示查询处理过程在磁盘活动的统计信息。（**实例位置：资源包\源码\11\11.20**）

SQL 语句如下：

```
USE db_2014
GO
SET STATISTICS IO ON
GO
SELECT Sname,Sex,Sage FROM Student WHERE Sex='男' AND Sage >20
GO
SET STATISTICS IO OFF;
GO
```

11.5.2　索引的维护

1．使用 DBCC SHOWCONTIG 语句

DBCC SHOWCONTIG 语句用来显示指定表的数据和索引的碎片信息。当对表进行大量的修改或

添加数据后，应该执行此语句来查看有无碎片。语法格式如下：

```
DBCC SHOWCONTIG
[(
    {table_name | table_id | view_name | view_id}
    [, index_name | index_id]
)]
    [WITH
        {
         [, [ALL_INDEXES]]
         [, [TABLERESULTS]]
         [, [FAST]]
         [, [ALL_LEVELS]]
         [NO_INFOMSGS]
        }
   ]
```

DBCC SHOWCONTIG 语句的参数及说明如表 11.3 所示。

表 11.3　DBCC SHOWCONTIG 语句的参数及说明

参　　数	描　　述
table_name | table_id | view_name | view_id	要检查碎片信息的表或视图。如果未指定，则检查当前数据库中的所有表和索引视图
index_name | index_id	要检查碎片信息的索引。如果未指定，则该语句将处理指定表或视图的基本索引
WITH	指定有关 DBCC 语句返回的信息类型的选项
FAST	指定是否要对索引执行快速扫描和输出最少信息。快速扫描不读取索引的叶级或数据级页
ALL_INDEXES	显示指定表和视图的所有索引的结果，即使指定了特定索引也是如此
TABLERESULTS	将结果显示为含附加信息的行集
ALL_LEVELS	仅为保持向后兼容性而保留
NO_INFOMSGS	取消严重级别从 0～10 的所有信息性消息

【例 11.21】　显示 db_2014 数据库中 Student 表的碎片信息，SQL 语句及运行结果如图 11.13 所示。（**实例位置：资源包\源码\11\11.21**）

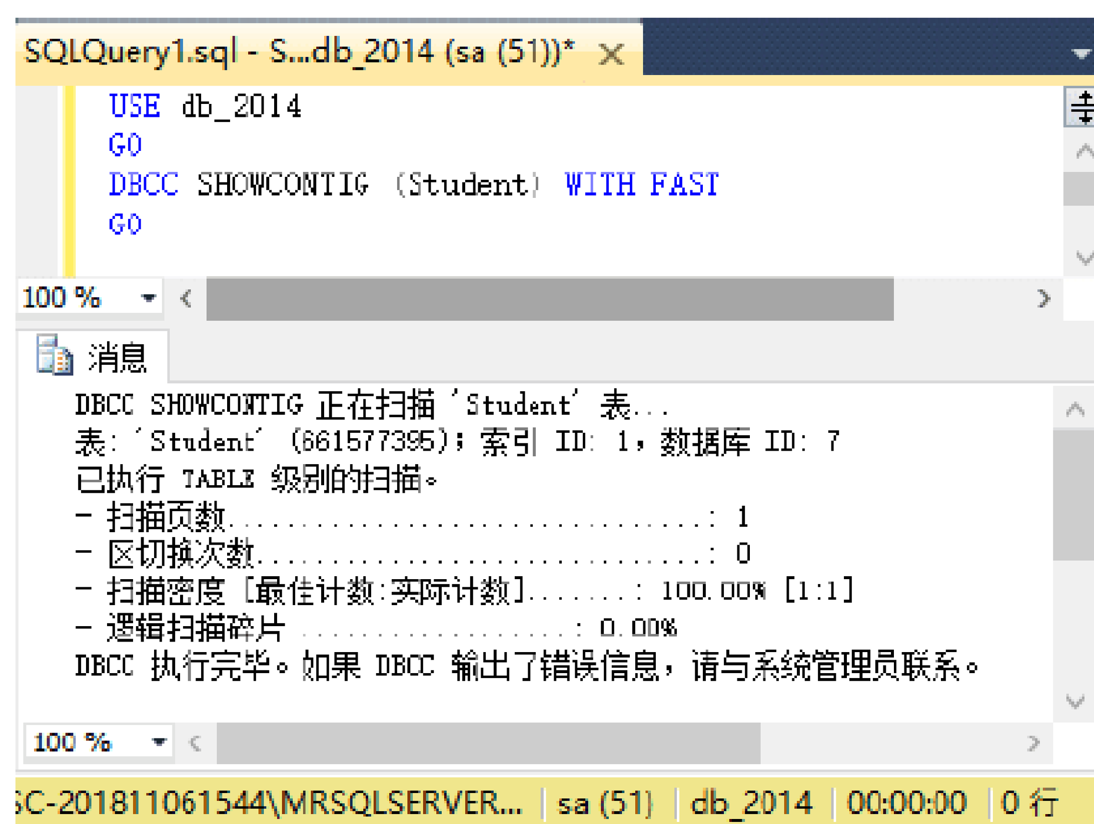
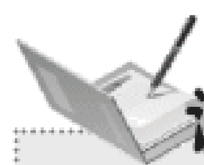


图 11.13　Student 表的碎片信息

SQL 语句如下：

```
USE db_2014
GO
DBCC SHOWCONTIG (Student) WITH FAST
GO
```



说明

当扫描密度为 100%时，说明表无碎片信息。

2．使用 DBCC DBREINDEX 语句

DBCC DBREINDEX 表示对指定数据库中的表重新生成一个或多个索引。语法格式如下：

```
DBCC DBREINDEX
(
    table_name
    [, index_name [, fillfactor]]
)
    [WITH NO_INFOMSGS]
```

参数说明如下。

☑ table_name：包含要重新生成的指定索引的表的名称。表名称必须遵循有关标识符的规则。

☑ index_name：要重新生成的索引名。索引名称必须符合标识符规则。

☑ fillfactor：在创建或重新生成索引时，每个索引页上用于存储数据的空间百分比。

☑ WITH NO_INFOMSGS：取消显示严重级别从 0～10 的所有信息性消息。

【例 11.22】 使用填充因子 100 重建 db_2014 数据库中 Student 表上的 MR_Stu_Sno 聚集索引。（**实例位置：资源包\源码\11\11.22**）

SQL 语句如下：

```
USE db_2014
GO
DBCC DBREINDEX('db_2014.dbo.Student',MR_Stu_Sno, 100)
GO
```

【例 11.23】 使用填充因子 100 重建 db_2014 数据库中 Student 表上的所有索引。（**实例位置：资源包\源码\11\11.23**）

SQL 语句如下：

```
USE db_2014
GO
DBCC DBREINDEX('db_2014.dbo.Student','',100)
GO
```

3．使用 DBCC INDEXDEFRAG 语句

DBCC INDEXDEFRAG 语句指定表或视图的索引碎片整理。语法格式如下：

```
DBCC INDEXDEFRAG
(
    {database_name | database_id | 0}
        , {table_name | table_id | view_name | view_id}
    [, {index_name | index_id} [, {partition_number | 0}]]
)
    [WITH NO_INFOMSGS]
```

DBCC INDEXDEFRAG 语句的参数及说明如表 11.4 所示。

表 11.4　DBCC INDEXDEFRAG 语句的参数及说明

参　　数	描　　述
database_name | database_id | 0	包含要进行碎片整理的索引的数据库。如果指定 0，则使用当前数据库
table_name | table_id | view_name | view_id	包含要进行碎片整理的索引的表或视图
index_name | index_id	要进行碎片整理的索引的名称或 ID。如果未指定，该语句将针对指定表或视图的所有索引进行碎片整理
partition_number | 0	要进行碎片整理的索引的分区号。如果未指定或指定 0，该语句将对指定索引的所有分区进行碎片整理
WITH NO_INFOMSGS	取消严重级别从 0～10 的所有信息性消息

【例 11.24】 清除数据库 db_2014 数据库中 Student 表的 MR_Stu_Sno 索引上的碎片。(**实例位置：资源包\源码\11\11.24**)

SQL 语句如下：

```
USE db_2014
GO
DBCC INDEXDEFRAG (db_2014,Student,MR_Stu_Sno)
GO
```



11.6　全 文 索 引

全文索引是一种特殊类型的基于标记的功能性索引，它是由 SQL Server 全文引擎生成和维护的。生成全文索引的过程不同于生成其他类型的索引。全文引擎并非基于特定行中存储的值来构造 B 树结构，而是基于要编制索引的文本中的各个标记来生成倒排、堆积且压缩的索引结构。

11.6.1　使用 SQL Server Management Studio 启用全文索引

操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 数据库。

（2）选择指定的数据库 db_2014，然后鼠标右键单击要创建索引的表，在弹出的快捷菜单中选择“全文检索”→“定义全文检索”命令，如图 11.14 所示。

（3）打开“全文索引向导”窗口，如图 11.15 所示。

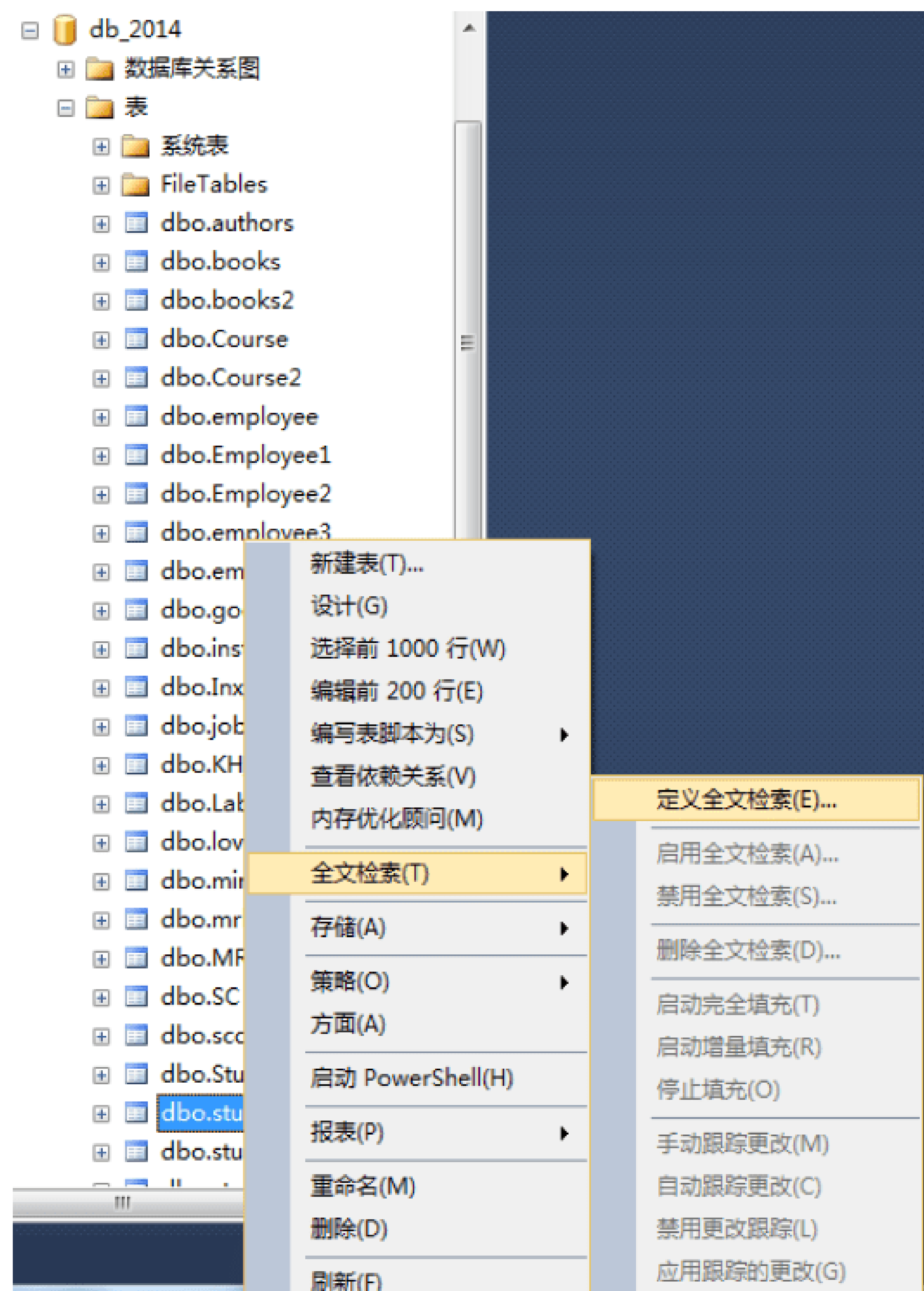


图 11.14　选择“定义全文检索”命令

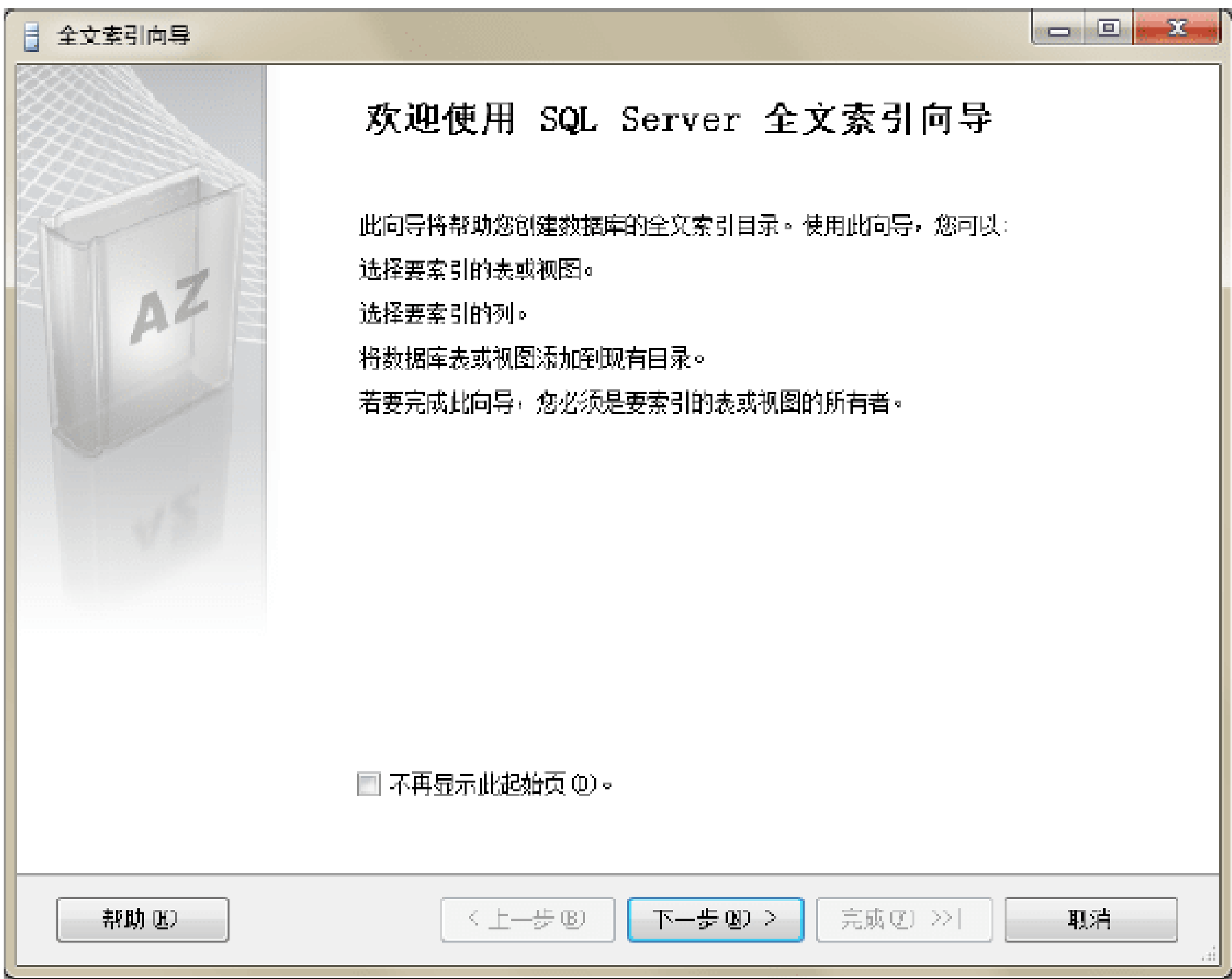


图 11.15　“全文索引向导”窗口

（4）单击“下一步”按钮，选择“唯一索引”，如图 11.16 所示。

（5）单击“下一步”按钮，选择表列，如图 11.17 所示。

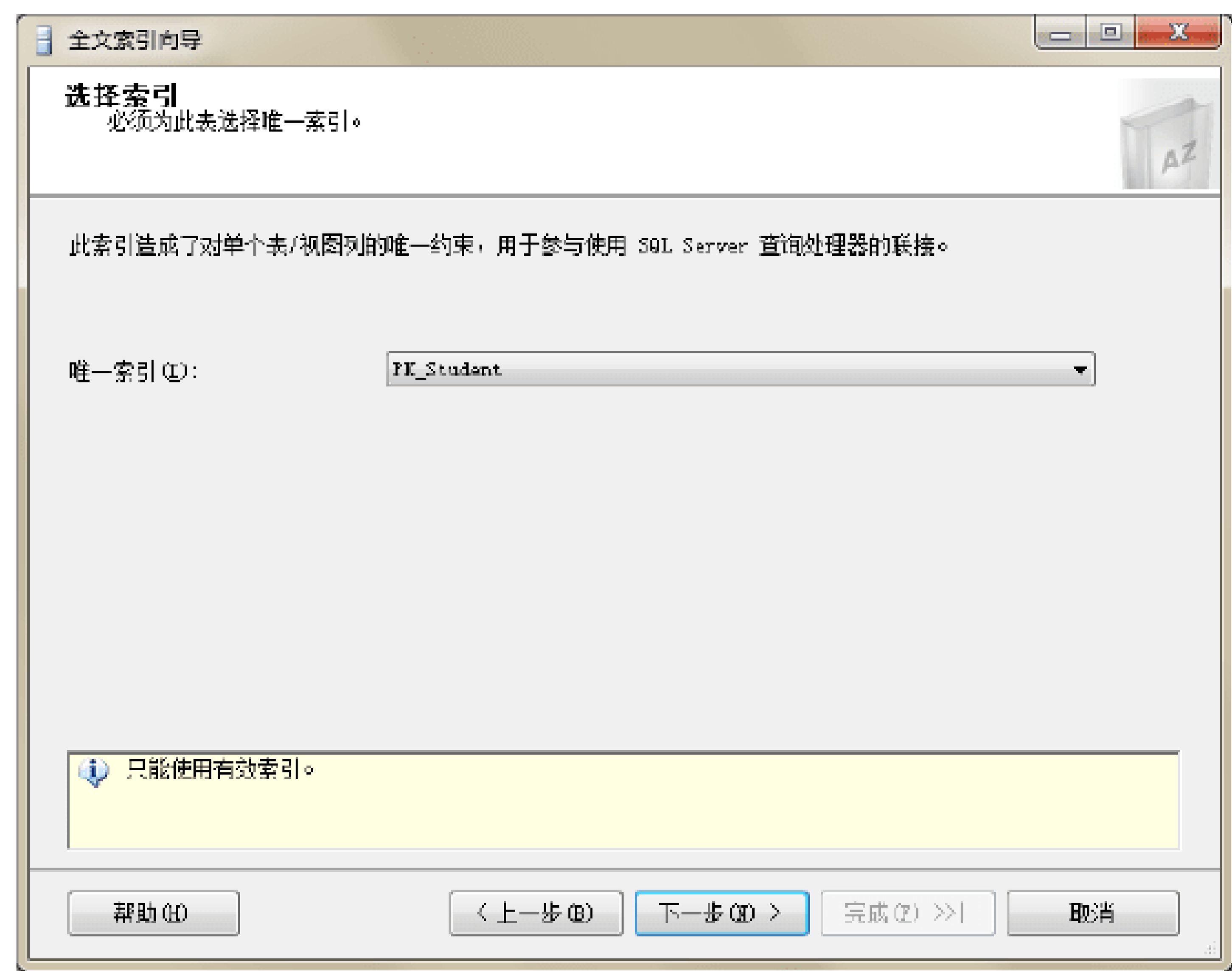


图 11.16　选择“唯一索引”

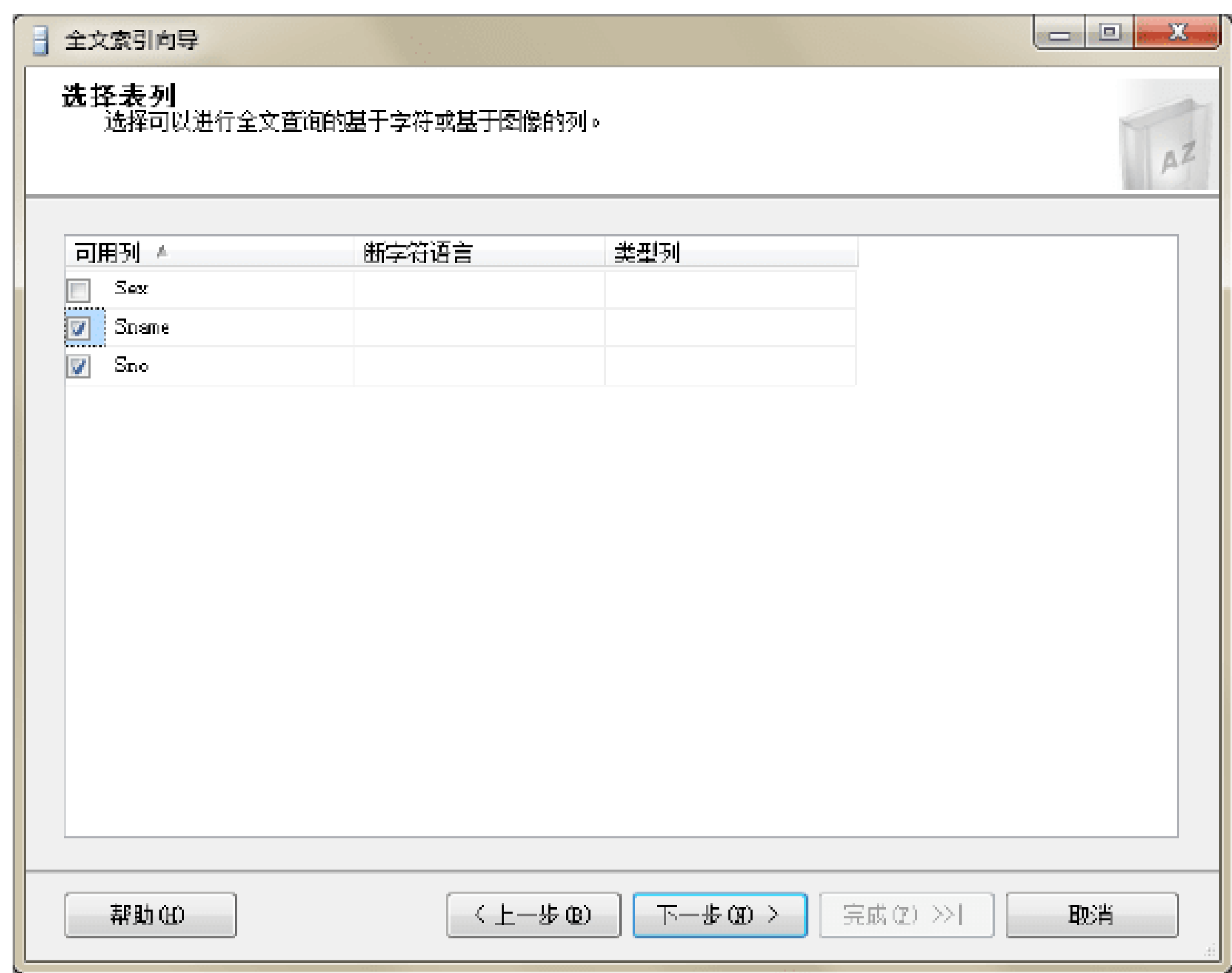


图 11.17　选择表列

（6）单击“下一步”按钮，选择跟踪表和视图更改的方式，如图 11.18 所示。

（7）单击“下一步”按钮，选中“创建新目录”复选框，在“名称”文本框中输入全文目录的名称，如图 11.19 所示。

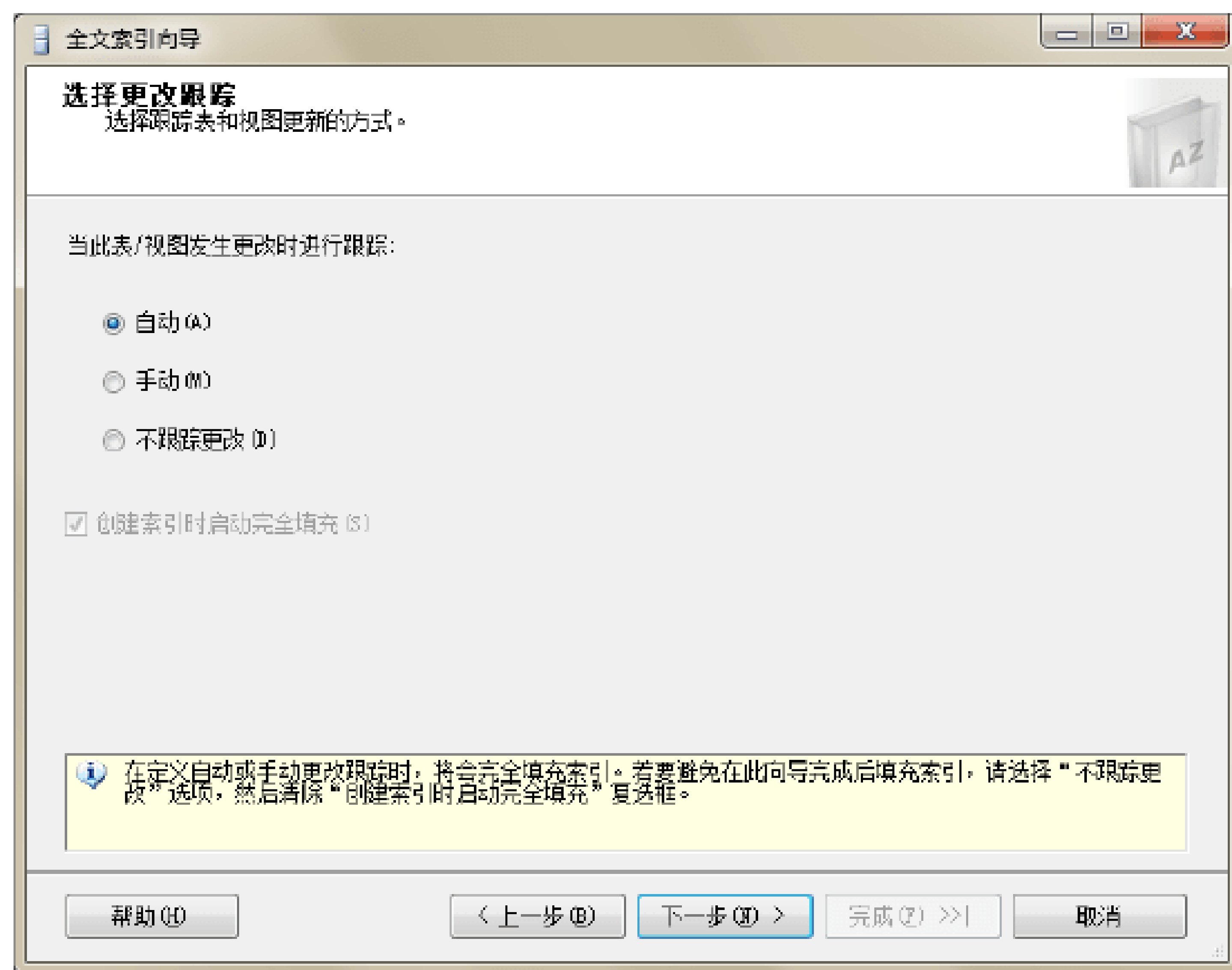


图 11.18　选择更改跟踪的方式

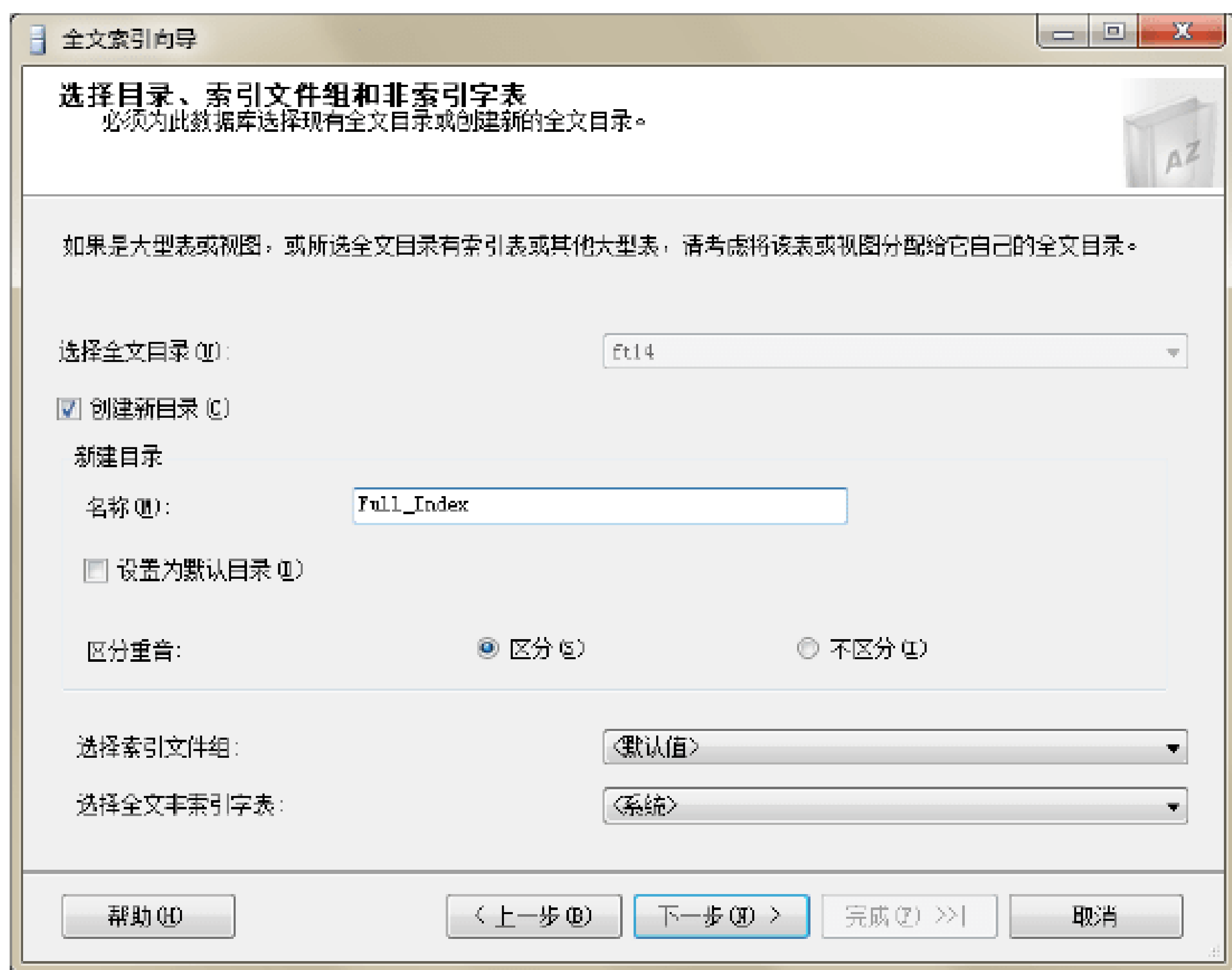


图 11.19　设置全文目录

（8）单击“下一步”按钮，弹出“定义填充计划（可选）”界面，如图 11.20 所示，此界面用来创建或修改此全文目录的填充计划（此计划是可选的）。在该界面中选择“新建表计划”或“新建目录计划”弹出新建计划的窗口，在新建窗口中输入计划的名称，设置执行的日期和时间，单击“确定”按钮即可。

（9）单击“下一步”按钮，弹出“全文索引向导说明”界面，如图 11.21 所示。

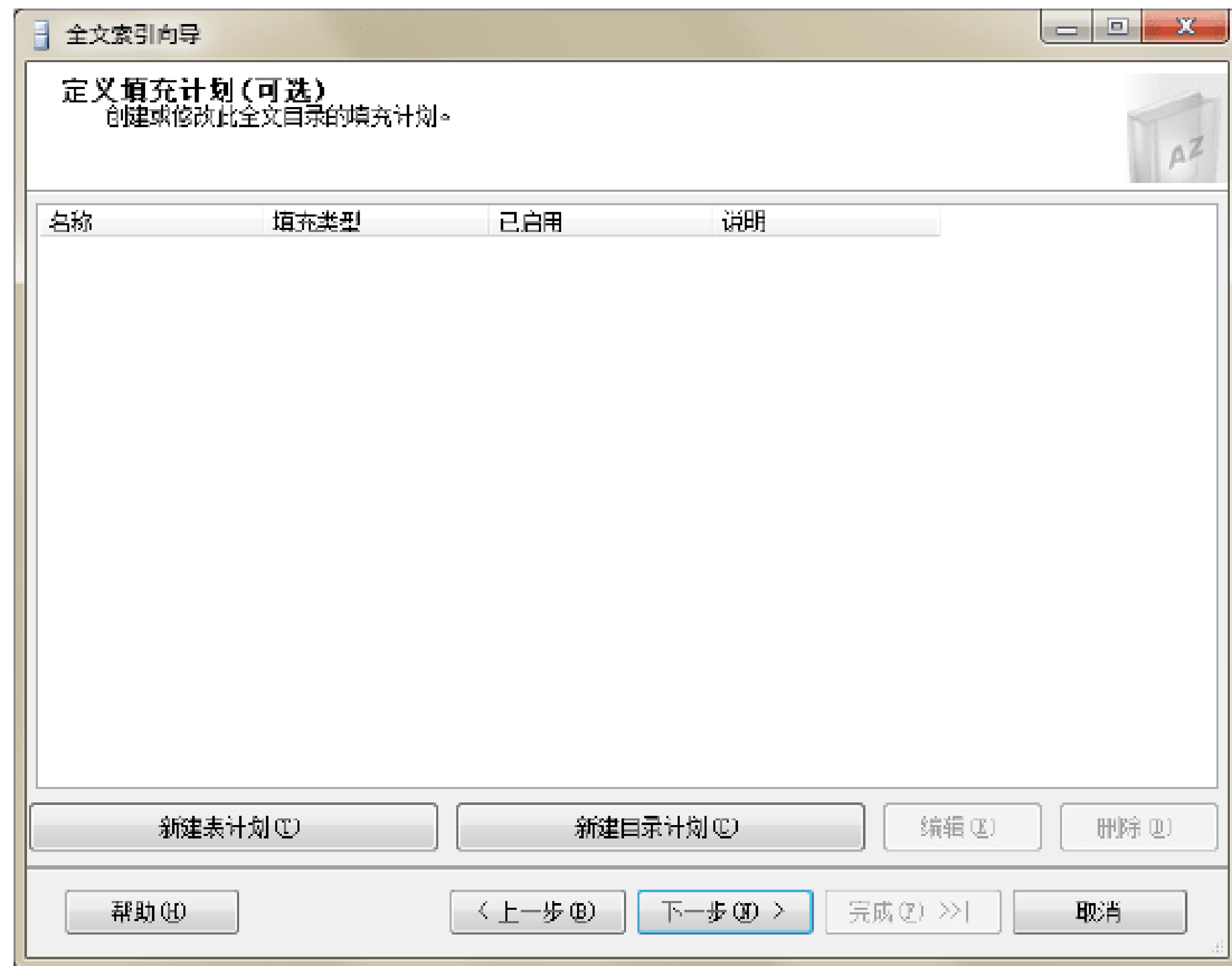


图 11.20　定义填充计划

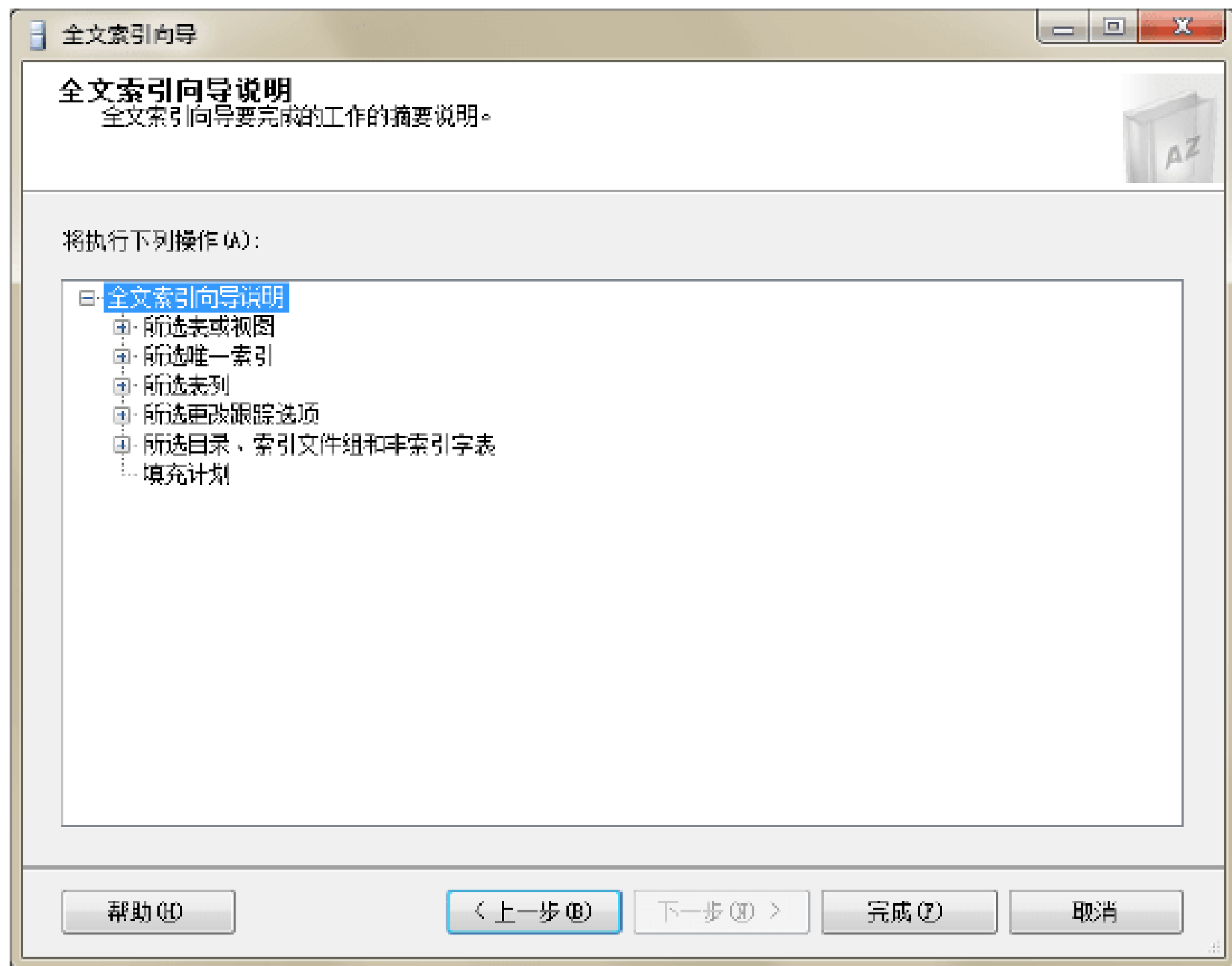


图 11.21　全文索引向导说明

（10）单击“完成”按钮，弹出“全文索引向导进度”界面，如图 11.22 所示。

（11）单击“关闭”按钮即可。

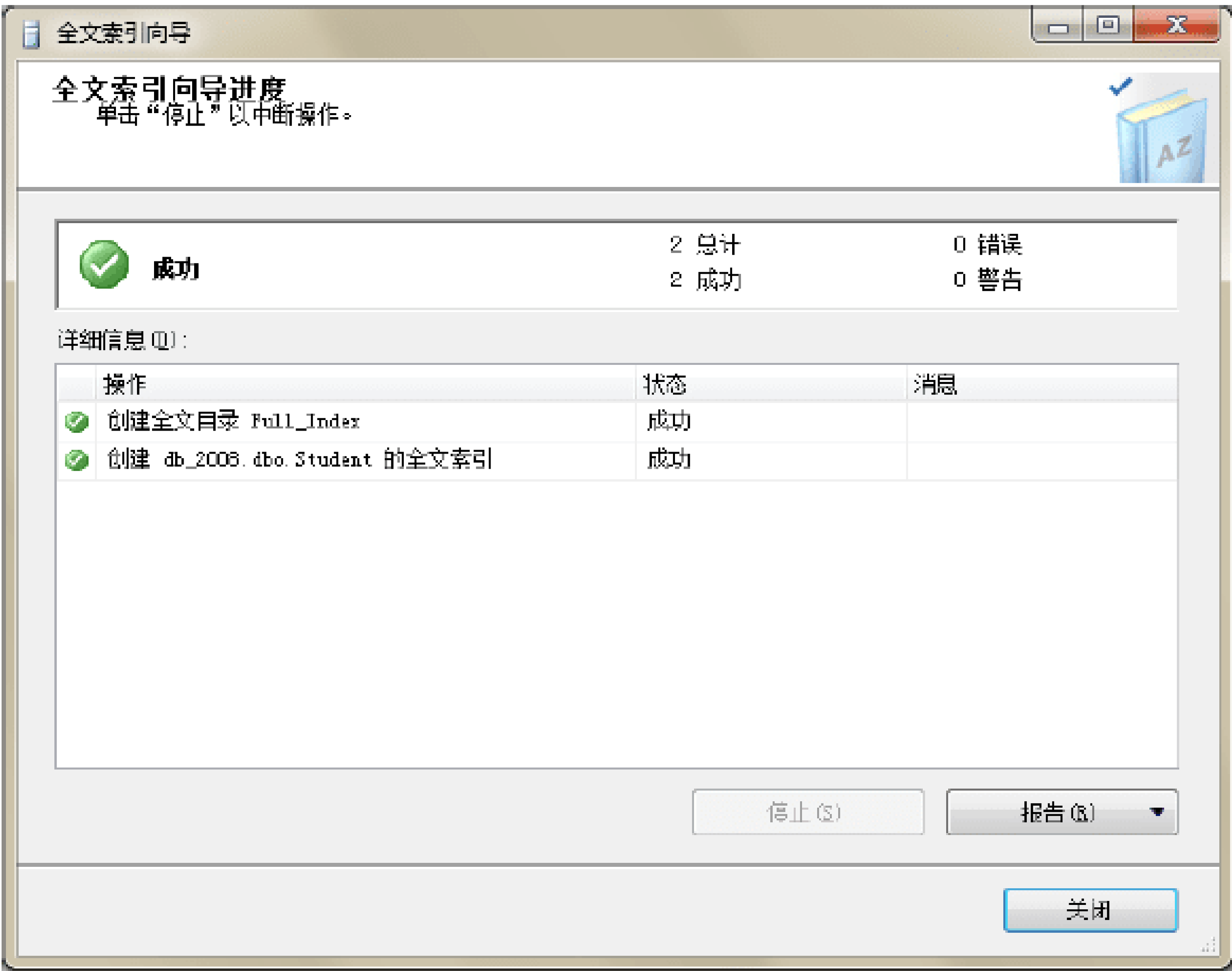


图 11.22　全文索引向导进度

11.6.2　使用 Transact-SQL 语句启用全文索引

1. 指定数据库启用全文索引

sp_fulltext_database 用于初始化全文索引，或者从当前数据库中删除所有的全文目录。在 SQL Server 2014 及更高版本中对全文目录无效，支持它仅仅是为了保持向后兼容。sp_fulltext_database 不会对给定数据库禁用全文引擎。在 SQL Server 2014 中，所有用户创建的数据库始终启用全文索引。

语法格式如下：

```
sp_fulltext_database [@action=] 'action'
```

参数[@action=] 'action'表示要执行的操作。action 的数据类型为 varchar(20)，参数取值如表 11.5 所示。

表 11.5　[@action =] 'action'参数的取值

值	描　述
enable	在当前数据库中启用全文索引
disable	对于当前数据库，删除文件系统中所有的全文目录，并且将该数据库标记为已经禁用全文索引。这个动作并不在全文目录或在表上更改任何全文索引元数据

【例 11.25】　使用数据库进行全文索引。（**实例位置：资源包\源码\11\11.25**）

SQL 语句如下：

```
USE db_2014
EXEC sp_fulltext_database 'enable'
```

运行结果如图 11.23 所示。

【例 11.26】　从数据库中删除全文索引。（**实例位置：资源包\源码\11\11.26**）

SQL 语句如下：

```
USE db_2014
EXEC sp_fulltext_database 'disable'
```

运行结果如图 11.24 所示。

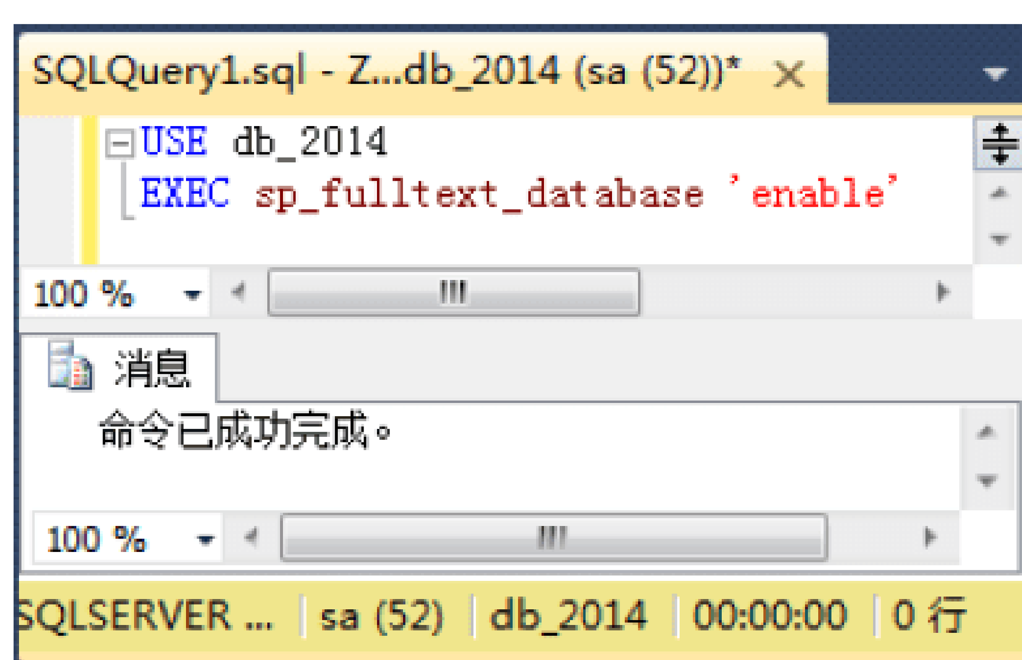


图 11.23　当前数据库启用全文索引

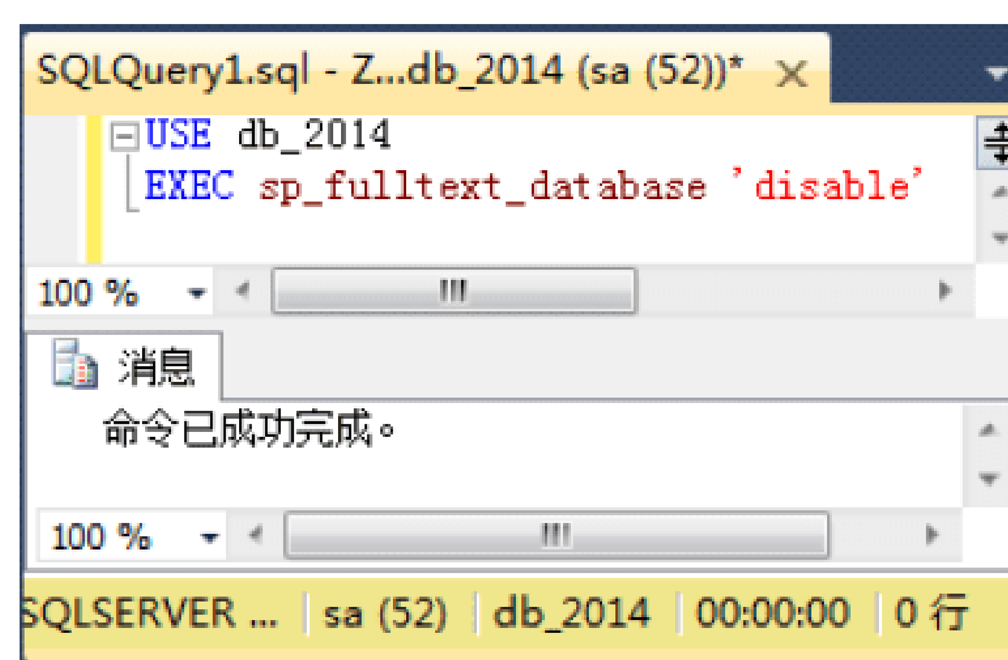


图 11.24　删除当前数据库的全文索引

2. 指定表启用全文索引

sp_fulltext_table 用于标记或取消标记要编制全文索引的表。
语法格式如下：

```
sp_fulltext_table [@tabname =] 'qualified_table_name'
    , [@action =] 'action'
    [, [@ftcat =] 'fulltext_catalog_name'
    , [@keyname =] 'unique_index_name']
```

参数说明如下。

☑ [@tabname =] 'qualified_table_name'：表名。该表必须存在当前的数据库中。数据类型为 nvarchar(517)，无默认值。
☑ [@action =] 'action'：将要执行的动作。action 的数据类型为 varchar(20)，无默认值，取值如表 11.6 所示。

表 11.6　[@action =] 'action'参数的取值

值	描　述
create	为 qualified_table_name 引用的表创建全文索引的元数据，并且指定该表的全文索引数据应该驻留在 fulltext_catalog_name 中
drop	除去全文索引上的元数据。如果全文索引是活动的，那么在除去它之前会自动停用它
activate	停用全文索引后，激活为 qualified_table_name 聚集全文索引的数据。在激活全文索引之前，应该至少有一列参与这个全文索引
deactivate	停用的全文索引，使得无法再为 qualified_table_name 聚集全文索引数据。全文索引元数据依然保留，并且该表还可以被重新激活
start_change_tracking	启动全文索引的增量填充。如果该表没有时间戳，那么就启动全文索引的完全填充，开始跟踪表发生的变化
stop_change_tracking	停止跟踪表发生的变化
update_index	将当前一系列跟踪的变化传播到全文索引

续表

值	描　述
start_background_updateindex	在变化发生时，开始将跟踪的变化传播到全文索引
stop_background_updateindex	在变化发生时，停止将跟踪的变化传播到全文索引
start_full	启动表的全文索引的完全填充
start_incremental	启动表的全文索引的增量填充

☑ [@ftcat =] 'fulltext_catalog_name'：create 动作有效的全文目录名。对于所有其他动作，该参数必须为 NULL。fulltext_catalog_name 的数据类型为 sysname，默认值为 NULL。

☑ [@keyname =] 'unique_index_name'：有效的单键列，create 动作在 qualified_table_name 上的唯一的非空索引。对于所有其他动作，该参数必须为 NULL。unique_index_name 的数据类型为 sysname，默认值为 NULL。

用表启用全文索引的操作步骤如下。

（1）将要启用全文索引的表创建一个唯一的非空索引（在以下示例中其索引名为 MR_Emp_ID_FIND）。

（2）用表所在的数据库启用全文索引。

（3）在该数据库中创建全文索引目录（在以下示例中全文索引目录为 ML_Employ）。

（4）用表启用全文索引标记。

（5）向表中添加索引字段。

（6）激活全文索引。

（7）启动完全填充。

【例 11.27】 创建一个全文索引标记，并在全文索引中添加字段。（**实例位置：资源包\源码\11\11.27**）

SQL 语句如下：

```
--将 Employee 表设为唯一索引
CREATE UNIQUE CLUSTERED INDEX MR_Emp_ID_FIND ON Employee (ID)
WITH IGNORE_DUP_KEY
--判断 db_2014 数据库是否可以创建全文索引
if (select DatabaseProperty('db_2014','IsFulltextEnabled'))=0
EXEC sp_fulltext_database 'enable'                                  --数据库启用全文索引
EXEC sp_fulltext_catalog 'ML_Employ','create'                       --创建全文索引目录为 ML_Employ
EXEC sp_fulltext_table 'Employee','create','ML_Employ','MR_Emp_ID_FIND'    --表启用全文索引标记
EXEC sp_fulltext_column 'Employee','Name','add'                     --添加全文索引字段
EXEC sp_fulltext_table 'Employee','activate'                        --激活全文索引
EXEC sp_fulltext_catalog 'ML_Employ','start_full'                   --启动表的全文索引的完全填充
```

11.6.3　使用 Transact-SQL 语句删除全文索引

DROP FULLTEXT INDEX 从指定的表或索引视图中删除全文索引。语法格式如下：

```
DROP FULLTEXT INDEX ON table_name
```

参数 table_name 表示包含要删除的全文索引的表或索引视图的名称。

【例 11.28】　删除 Employee 数据表的全文索引 MR_Emp_ID_FIND。（**实例位置：资源包\源码\11\11.28**）

SQL 语句如下：

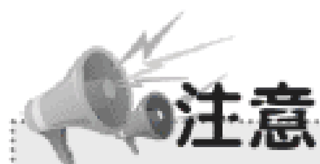
```
USE db_2014
DROP FULLTEXT INDEX ON Employee
```

11.6.4　全文目录

对于 SQL Server 2014 数据库，全文目录为虚拟对象，并不属于任何文件组；它是一个表示一组全文索引的逻辑概念。

1．全文目录的创建、删除和重创建

sp_fulltext_catalog 用于创建和删除全文目录，并启动和停止目录的索引操作。可为每个数据库创建多个全文目录。



注意

在 Microsoft SQL Server 2008 之后将删除该功能。请避免在新的开发工作中使用该功能，并着手修改当前还在使用该功能的应用程序。

语法格式如下：

```
sp_fulltext_catalog [@ftcat =] 'fulltext_catalog_name' ,
    [@action =] 'action'
    [, [@path =] 'root_directory']
```

参数说明如下。

☑　[@ftcat =] 'fulltext_catalog_name'：全文目录的名称。对于每个数据库，目录名必须是唯一的。其数据类型为 sysname。

☑　[@action =] 'action'：将要执行的动作。action 的数据类型为 varchar(20)，取值如表 11.7 所示。

表 11.7　[@action =] 'action'参数的取值

值	描　　述
create	在文件系统中创建一个空的新全文目录，并向 sysfulltextcatalogs 添加一行
drop	将全文目录从文件系统中删除，并且删除 sysfulltextcatalogs 中相关的行
start_incremental	启动全文目录的增量填充。如果目录不存在，就会显示错误
start_full	启动全文目录的完全填充。即使与此全文目录相关联的每一个表的每一行都进行过索引，也会对其检索全文索引
stop	停止全文目录的索引填充。如果目录不存在，就会显示错误。如果已经停止了填充，那么并不会显示警告
rebuild	重建全文目录，方法是从文件系统中删除现有的全文目录，然后重建全文目录，并使该全文目录与所有带有全文索引引用的表重新建立关联

【例 11.29】　创建一个空的全文目录 QWML。（**实例位置：资源包\源码\11\11.29**）

SQL 语句如下：

```
USE db_2014
GO
EXEC sp_fulltext_database 'enable'      --数据库启用全文索引
EXEC sp_fulltext_catalog 'QWML','create'
```

【例 11.30】　重新创建一个已有的全文目录 QWML。（**实例位置：资源包\源码\11\11.30**）

SQL 语句如下：

```
USE db_2014
GO
EXEC sp_fulltext_database 'enable'      --数据库启用全文索引
EXEC sp_fulltext_catalog 'QWML','rebuild'
```

运行结果如图 11.25 所示。

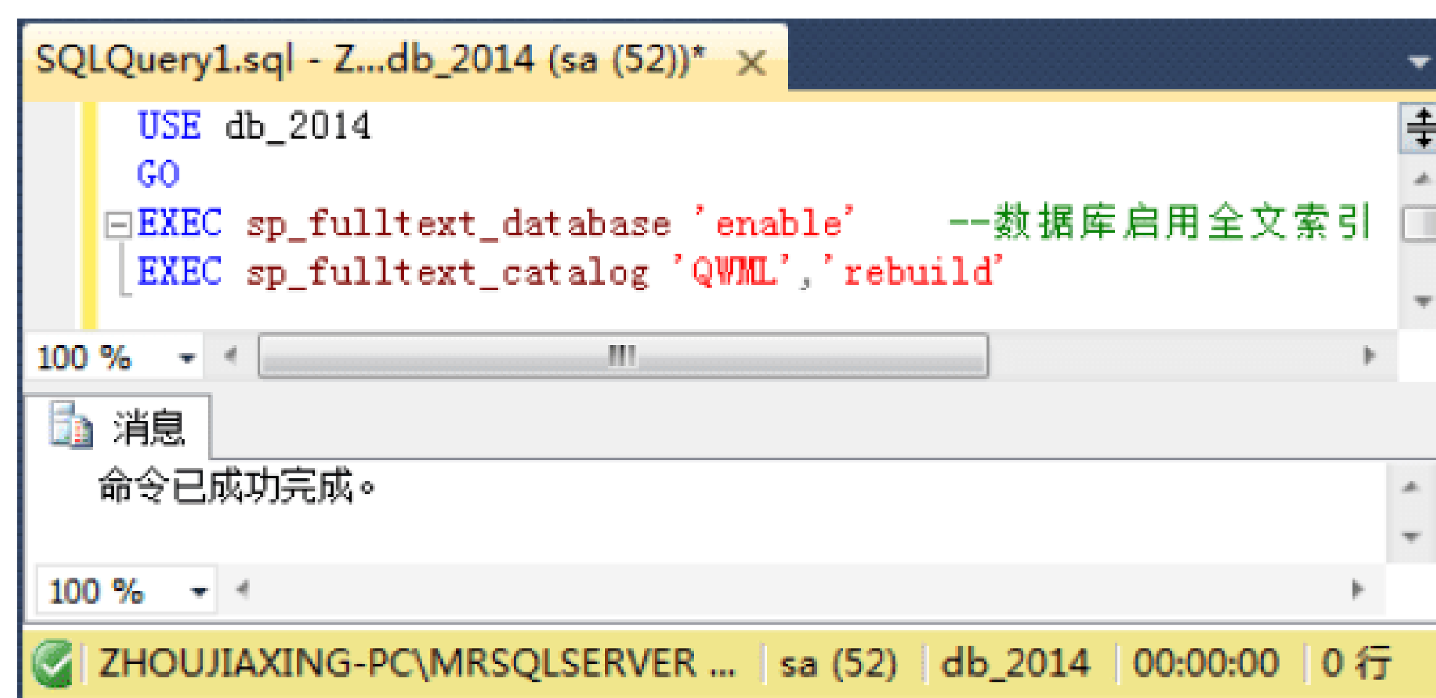


图 11.25　重建一个全文目录

【例 11.31】　删除全文目录 QWML。（**实例位置：资源包\源码\11\11.31**）

SQL 语句如下：

```
USE db_2014
GO
EXEC sp_fulltext_catalog 'QWML','drop'
```

2．向全文目录中增加、删除列

sp_fulltext_column 指定表的某个特定列是否参与全文索引。



注意

在 Microsoft SQL Server 2005 之后将删除该功能。请避免在新的开发工作中使用该功能，并着手修改当前还在使用该功能的应用程序。

语法格式如下：

```
sp_fulltext_column [@tabname=] 'qualified_table_name' ,
     [@colname=] 'column_name' ,
```

```
    [@action=] 'action'
    [, [@language=] 'language_term']
    [, [@type_colname=] 'type_column_name']
```

参数说明如下。

- ☑ [@tabname=] 'qualified_table_name'：由一部分或两部分组成的表的名称。表必须在当前数据库中。表必须有全文索引。qualified_table_name 的数据类型为 nvarchar(517)，无默认值。
- ☑ [@colname=] 'column_name'：qualified_table_name 中列的名称。列必须为字符列、varbinary(max)列或 image 列，不能是计算列。column_name 的数据类型为 sysname，无默认值。



注意

SQL Server 可以为存储在数据类型为 varbinary(max)或 image 的列中的文本数据创建全文索引。不对图像和图片进行索引。

- ☑ [@action=] 'action'：要执行的操作。action 的数据类型为 varchar(20)，无默认值，可以是表 11.8 中的列值之一。

表 11.8　[@action =] 'action'参数的取值

值	描　述
add	将 qualified_table_name 的 column_name 添加到表的非活动全文索引中。该动作启用全文索引的列
drop	从表的非活动全文索引中删除 qualified_table_name 的 column_name

- ☑ [@language=] 'language_term'：存储在列中的数据的语言。
- ☑ [@type_colname =] 'type_column_name'：qualified_table_name 中列的名称，用于保存 column_name 的文档类型。此列必须是 char、nchar、varchar 或 nvarchar。仅当 column_name 数据类型为 varbinary(max)或 image 时才使用该列。type_column_name 的数据类型为 sysname，无默认值。

【例 11.32】　将 Student 表的 Sex 列添加到表的全文索引。(**实例位置：资源包\源码\11\11.32**)

SQL 语句如下：

```
USE db_2014
EXEC sp_fulltext_column Student, Sex, 'add'
```

运行结果如图 11.26 所示。

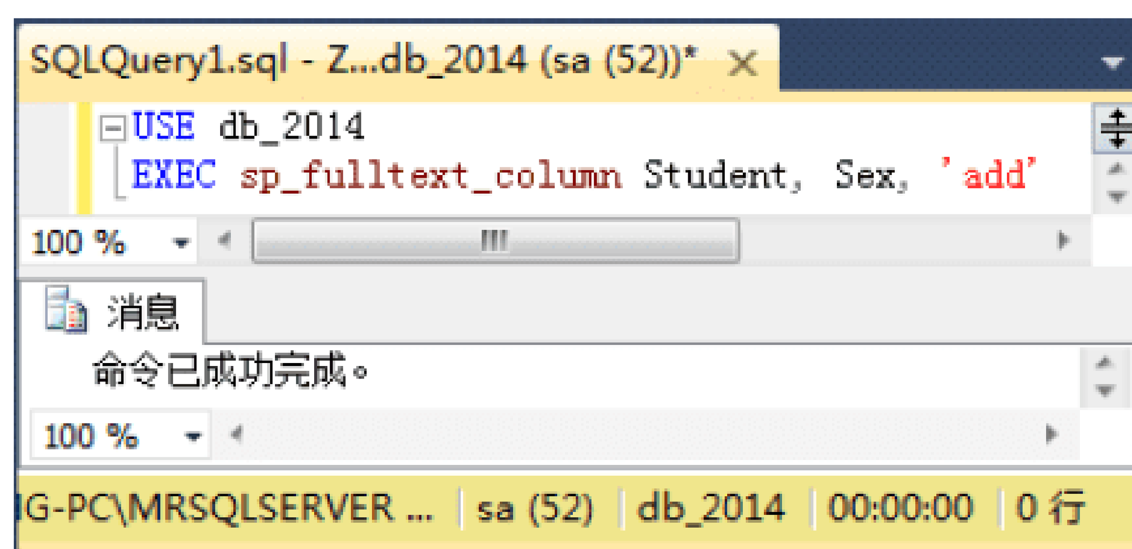


图 11.26　在列中添加表的全文索引

【例 11.33】　将 Student 表的 Sex 列从全文索引中删除。(**实例位置：资源包\源码\11\11.33**)

SQL 语句如下：

```
USE db_2014
EXEC sp_fulltext_column Student, Sex, 'drop'
```

3．激活全文目录

要激活表 Student 的全文目录，首先要在表中创建全文索引。

【例 11.34】　激活 Employee 表中的全文目录。（**实例位置：资源包\源码\11\11.34**）

SQL 语句如下：

```
USE db_2014
EXEC sp_fulltext_table 'Employee','activate'
```

这样就完成了对全文目录的定义，如果要对创建的全文目录进行初始化填充，可以使用如下 SQL 语句：

```
USE db_2014
EXEC sp_fulltext_table 'Employee','start_full'
```

填充也称为爬网，是创建和维护全文索引的过程。

11.6.5　全文目录的维护

1．用 SQL Server Management Studio 来维护全文目录

操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 数据库。

（2）选择指定数据库中的数据表（这里以 db_2014 数据库中的 Employee 表为例，该表已经创建全文索引）。

（3）在 Employee 表上单击鼠标右键，在弹出的快捷菜单中选择“全文索引”命令，如图 11.27 所示。

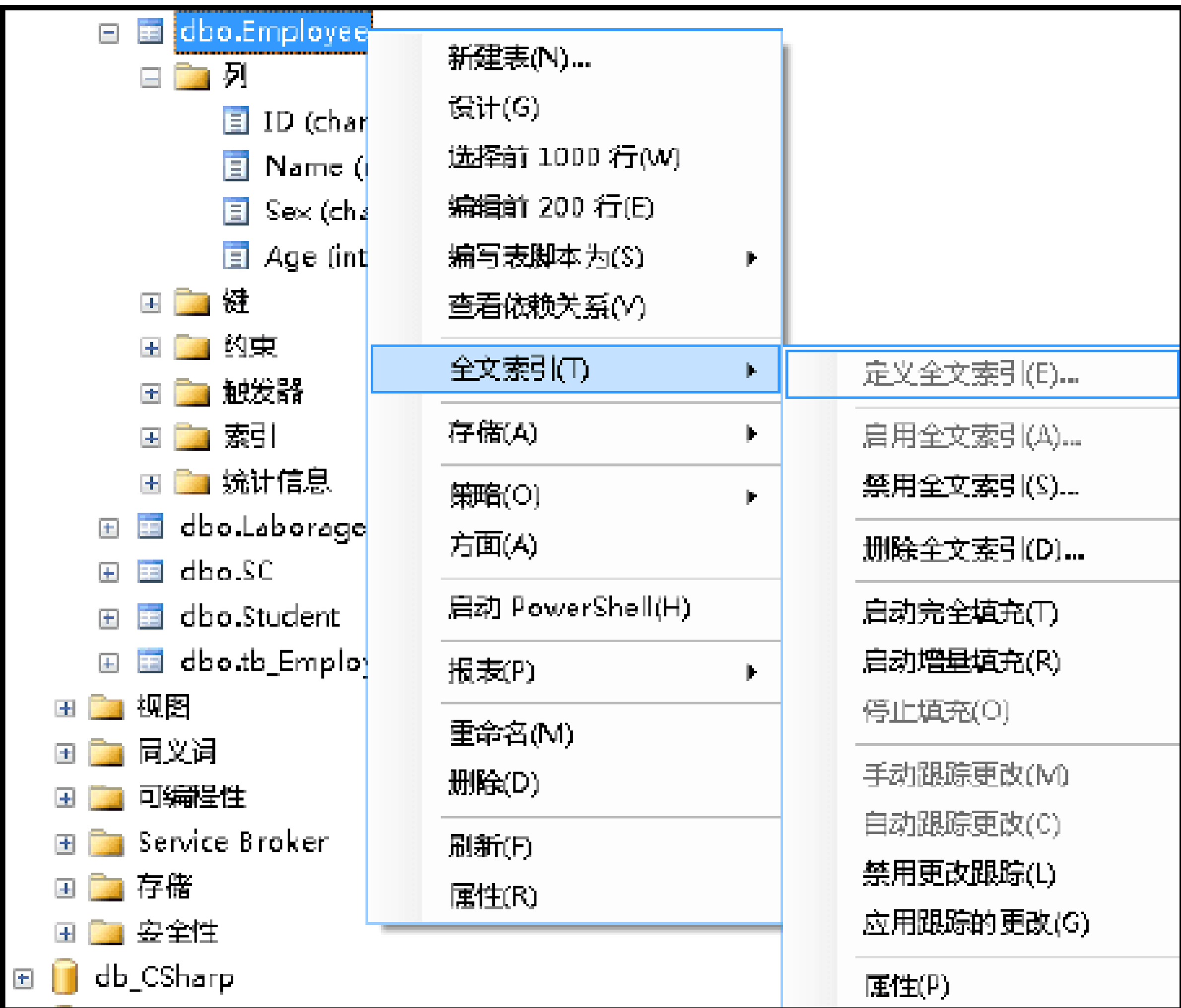


图 11.27　维护全文目录

（4）在“全文索引”的子菜单中可以对全文目录进行修改，具体功能如表 11.9 所示。

表 11.9　维护全文目录

选　项	描　述
删除全文索引	将选定的表从它的全文目录中删除
启动完全填充	使用选定表中的全部行对全文目录进行初始的数据填充
启动增量填充	识别选定的表从最后一次填充所发生的数据变化，并利用最后一次添加、删除或修改的行对全文索引进行填充
停止填充	终止当前正在运行的全文索引填充任务
手动跟踪更改	手动的方式使应用程序可以仅获取对用户表所做的更改以及与这些更改有关的信息
自动跟踪更改	自动使应用程序可以仅获取对用户表所做的更改以及与这些更改有关的信息
禁用更改跟踪	不让应用程序获取对用户表所做的更改以及与这些更改有关的信息
应用跟踪的更改	应用应用程序获取对用户表所做的更改及与这些更改有关的信息

2．使用 Transact-SQL 语句维护全文目录

以 Employee 表为例介绍如何使用 Transact-SQL 语句维护全文目录，Employee 为已经创建全文索引的数据表。

（1）完全填充

```
EXEC sp_fulltext_table 'Employee','start_full'
```

（2）增量填充

```
EXEC sp_fulltext_table 'Employee','start_incremental'
```

（3）更改跟踪

```
EXEC sp_fulltext_table ' Employee ','start_change_tracking'
```

（4）后台更新

```
EXEC sp_fulltext_table ' Employee ','start_background_updateindex'
```

（5）清除无用的全文目录

```
EXEC sp_fulltext_service 'clean_up'
```

（6）sp_help_fulltext_catalogs

返回指定的全文目录的 ID（ftcatid）、名称（NAME）、根目录（PATH）、状态（STATUS）以及全文索引表的数量（NUMBER_FULLTEXT_TABLES）。

【例 11.35】　返回有关全文目录 QWML 的信息。（**实例位置：资源包\源码\11\11.35**）

SQL 语句如下：

```
USE db_2014
GO
```

```
EXEC sp_help_fulltext_catalogs 'QWML';
GO
```

运行结果如图 11.28 所示。

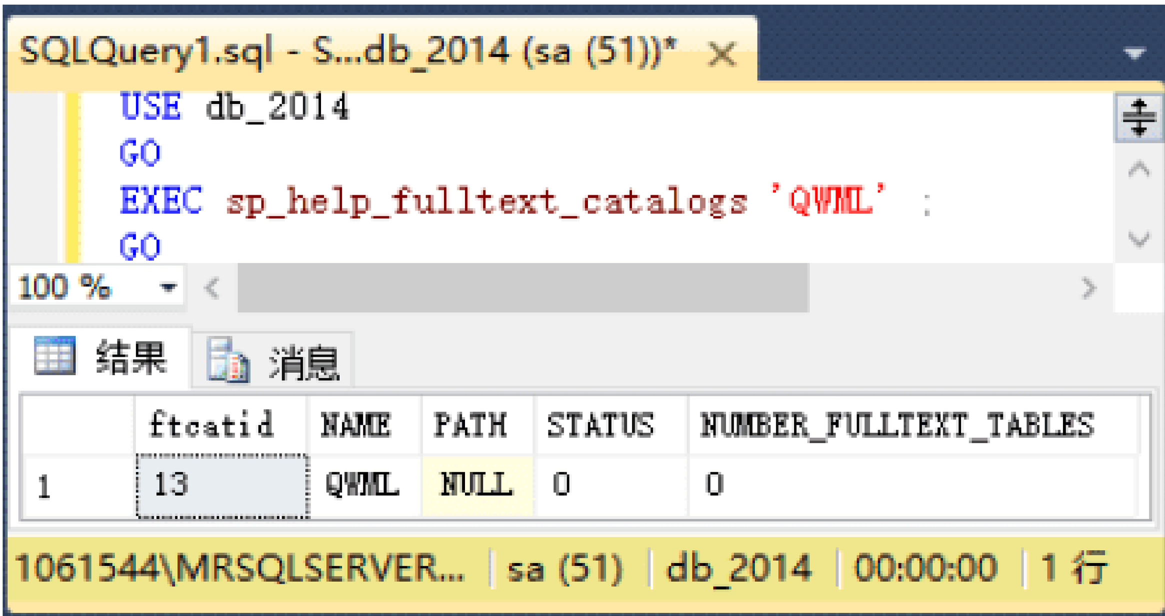


图 11.28　返回全文目录 QWML 的信息

STATUS 列将返回指定全文目录的当前状态，如表 11.10 所示。

表 11.10　STATUS 列的返回状态

返　回　值	描　　述	返　回　值	描　　述
0	空闲	5	关闭
1	正在进行完全填充	6	正在进行增量填充
2	暂停	7	生成索引
3	已中止	8	磁盘已满，已暂停
4	正在恢复	9	更改跟踪

（7）sp_help_fulltext_tables

该存储过程返回为全文索引注册的表的列表。

【例 11.36】　返回包含在指定全文目录 QWML 中的表的信息。(**实例位置：资源包\源码\11\11.36**)

SQL 语句如下：

```
USE db_2014
EXEC sp_help_fulltext_tables 'QWML'
```

运行结果如图 11.29 所示。

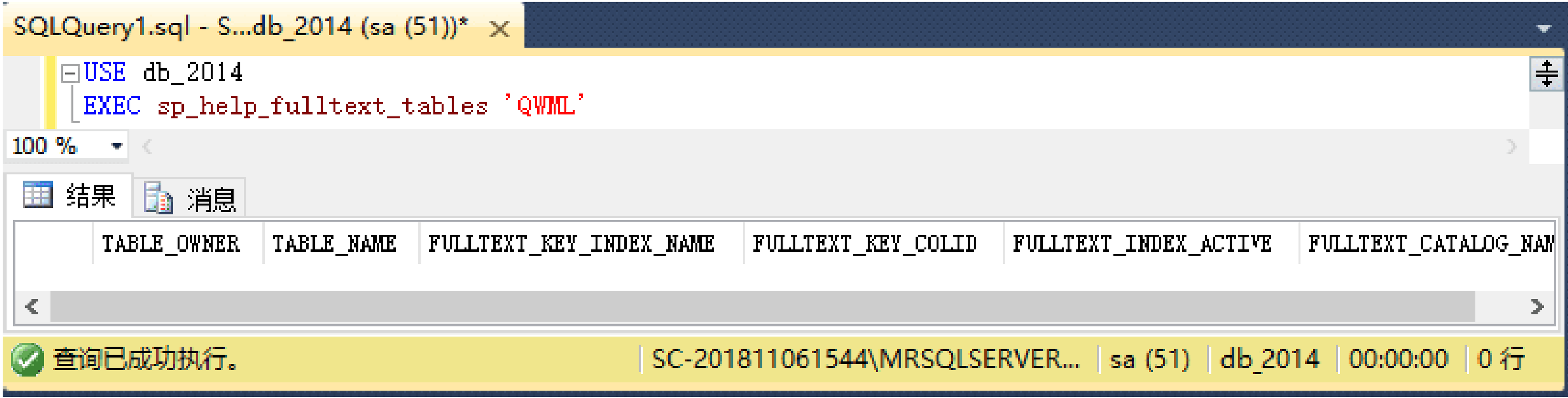


图 11.29　返回包含在指定全文目录

（8）sp_help_fulltext_columns

该存储过程返回为全文索引指定的列。

【例 11.37】　返回 Inx_table 表中全文索引，Inx_table 表为已创建全文索引的数据表。(**实例位置：资源包\源码\11\11.37**)

SQL 语句如下：

```
USE db_2014
EXEC sp_help_fulltext_columns 'Inx_table'
```

运行结果如图 11.30 所示。

	TABLE_OWNER	TABLE_ID	TABLE_NAME	FULLTEXT_COLUMN_NAME	FULLTEXT_COLID
1	dbo	1282103608	Inx_table	Name	2

图 11.30　返回全文索引指定的列



11.7　数据完整性

数据完整性是 SQL Server 用于保证数据库中数据一致性的一种机制，防止非法数据存入数据库。具体的数据完整性主要体现在以下几点。

☑　数据类型准确无误。

☑　数据取值符合规定的范围。

☑　多个数据表之间的数据不存在冲突。

下面介绍 SQL Server 2014 提供的 4 种数据完整性机制：域完整性、实体完整性、引用完整性和用户定义完整性。

11.7.1　域完整性

域是指数据表中的列（字段），域完整性就是指列的完整性。实现域完整性的方法有：限制类型（通过数据类型）、格式（通过 CHECK 约束和规则）或可能的取值范围（通过 CHECK 约束、DEFAULT 定义、NOT NULL 定义和规则）等，它要求数据表中指定列的数据具有正确的数据类型、格式和有效的数据范围。

域完整性常见的实现机制包括以下方面。

☑　默认值（Default）

☑　检查（Check）

☑　外键（Foreign Key）

☑　数据类型（Data Type）

☑　规则（Rule）

【例 11.38】　创建表 student2，有学号、最好成绩和平均成绩 3 列，求最好成绩必须大于平均成绩。（**实例位置：资源包\源码\11\11.38**）

SQL 语句如下：

```
CREATE TABLE student2
(
学号  char(6) not null,
最好成绩  int not null,
平均成绩  int not null,
CHECK(最好成绩>平均成绩)
)
```

运行结果如图 11.31 所示。

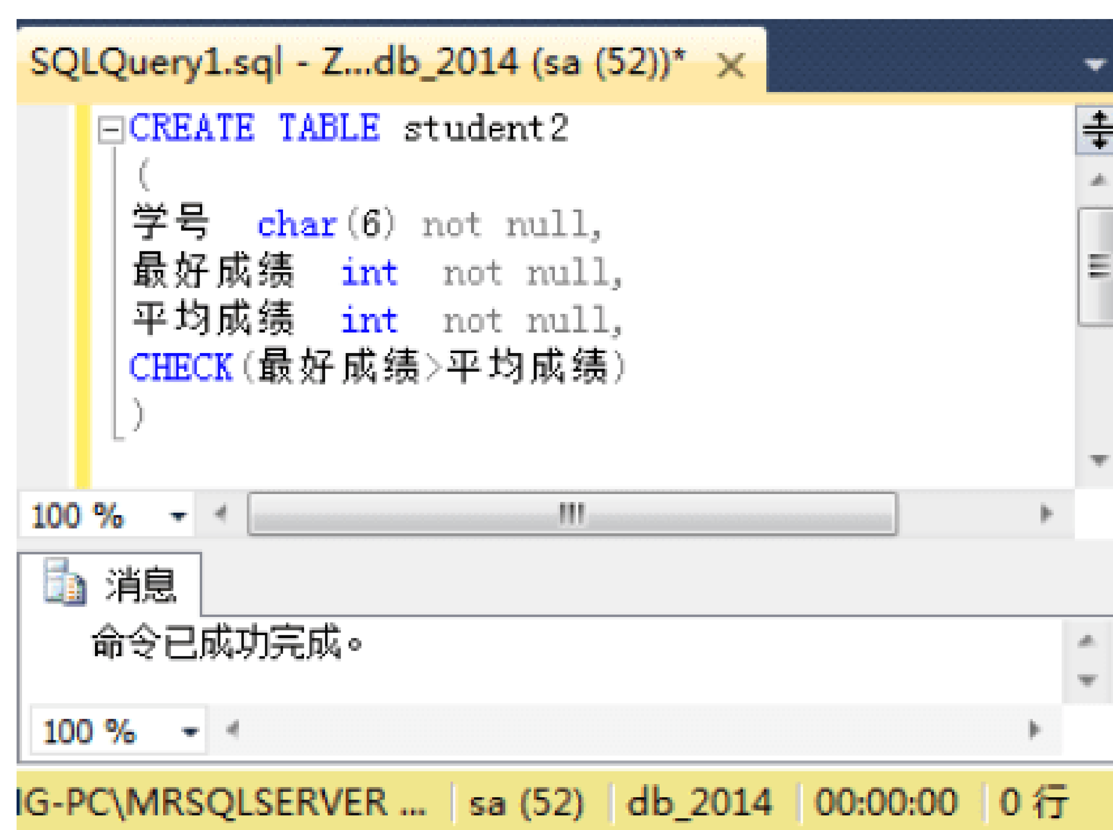


图 11.31　域完整性

11.7.2　实体完整性

现实世界中，任何一个实体都有区别于其他实体的特征，即实体完整性。在 SQL Server 数据库中，实体完整性是指所有的记录都应该有一个唯一的标识，以确保数据表中数据的唯一性。

如果将数据库中数据表的第一行看作一个实体，可以通过以下几项实施实体完整性。

☑ 唯一索引（Unique Index）

☑ 主键（Primary Key）

☑ 唯一码（Unique Key）

☑ 标识列（Identity Column）

【例 11.39】 创建表 student3，并对借书证号字段创建 PRIMARY KEY 约束，对姓名字段定义 UNIQUE 约束。（**实例位置：资源包\源码\11\11.39**）

SQL 语句如下：

```
USE db_2014
Go
CREATE TABLE student3
(
借书证号  char(8) not null CONSTRAINT py PRIMARY KEY,
姓名  char(8) not null CONSTRAINT uk UNIQUE,
专业  char(12) not null,
性别  bit not null,
```

```
借书量 int CHECK(借书量>=0 AND 借书量<=20) null
)
go
```

运行结果如图 11.32 所示。

【例 11.40】　创建表 student4，由借书证号、索书名、借书时间作为联合主键。（**实例位置：资源包\源码\11\11.40**）

SQL 语句如下：

```
Use db_2014
CREATE TABLE student4
(
 借书证号 char(8) not null,
 索书名 char(10) not null,
 借书时间 date not null,
 还书时间 date not null,
PRIMARY KEY(索书名,借书证号,借书时间)
)
```

运行结果如图 11.33 所示。

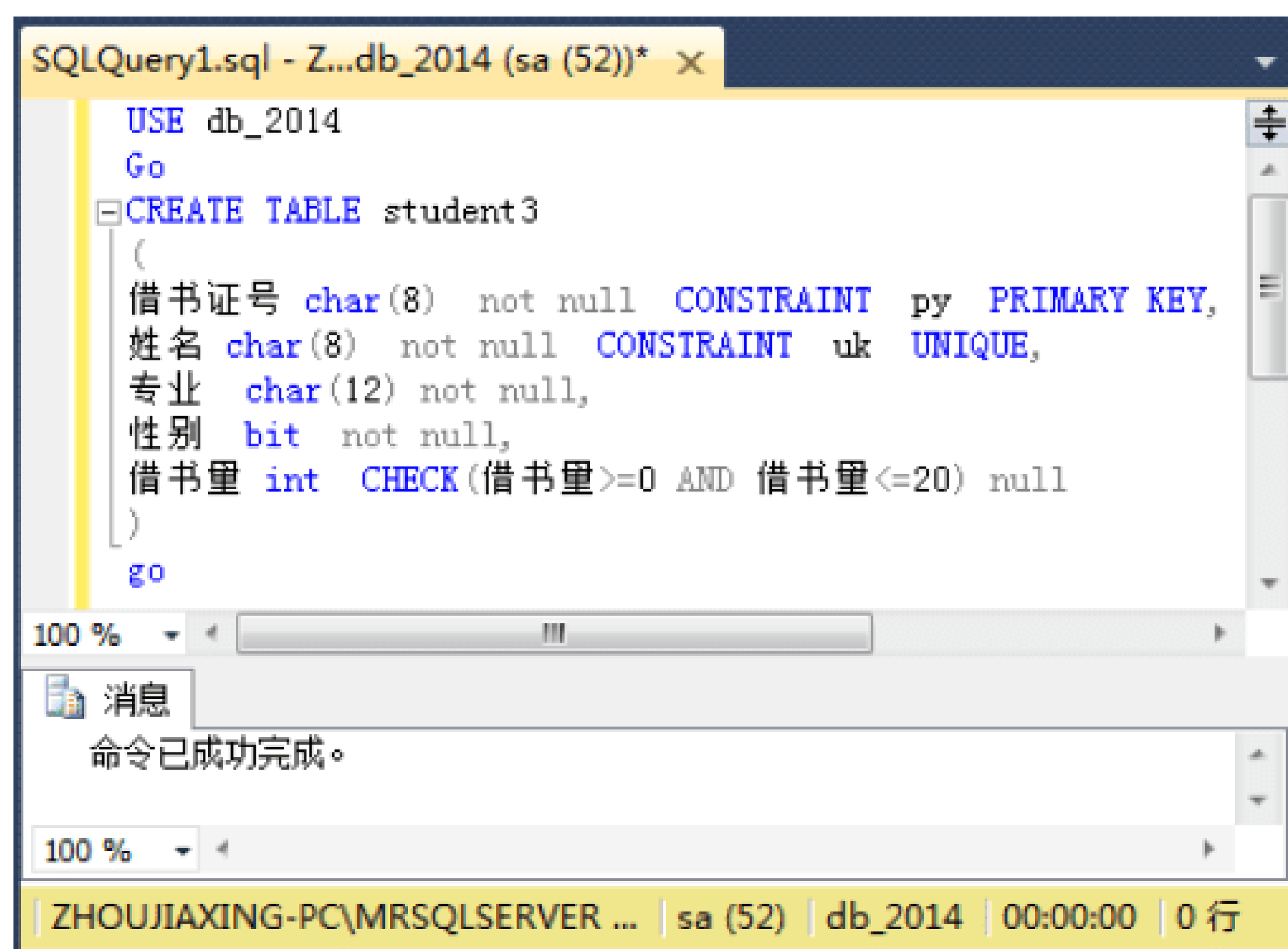


图 11.32　实体完整性设置

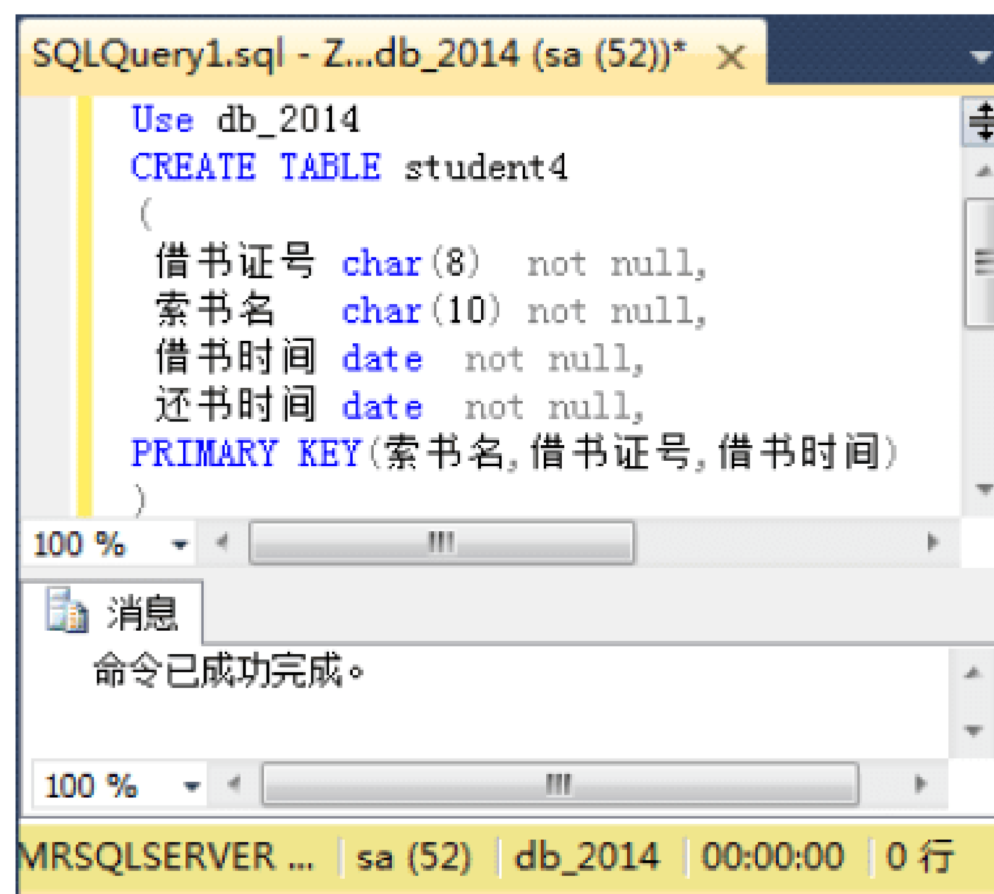


图 11.33　联合主键

11.7.3　引用完整性

引用完整性又称参照完整性，引用完整性保证主表中的数据与从表中数据的一致性。在 SQL Server 2014 中，参照完整性的实现是通过定义外键与主键之间或外键与唯一键之间的对应关系实现的。引用完整性确保键值在所有表中一致。引用完整性的实现方法如下。

☑　外键（Foreign Key）
☑　检查（Check）
☑　触发器（Trigger）
☑　存储过程（Stored Procedure）

【例 11.41】　创建表 student5，要求表中所用的索书名、借书证号和借书时间组合都必须出现在

student4 表中。（**实例位置：资源包\源码\11\11.41**）

SQL 语句如下：

```
Use db_2014
CREATE TABLE student5
(
借书证号  char(8) NOT NULL,
ISBN char(16) NOT NULL,
索书名  char(10) NOT NULL,
借书时间  date NOT NULL,
还书时间  date NOT NULL,
CONSTRAINT FK_point FOREIGN KEY (索书名,借书证号,借书时间)
REFERENCES student4 (索书名,借书证号,借书时间)
ON DELETE NO ACTION
)
```

运行结果如图 11.34 所示。

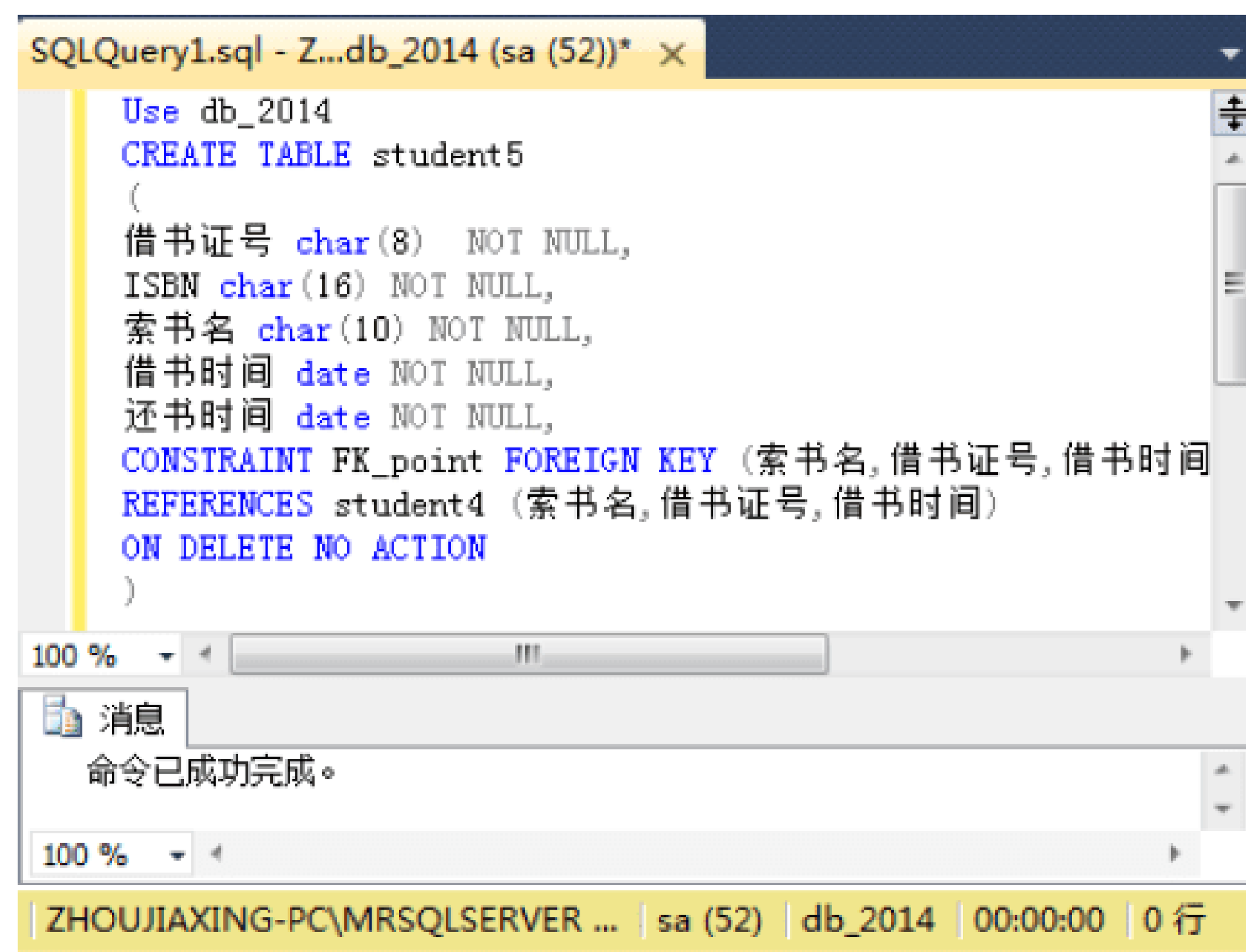


图 11.34　引用完整性

11.7.4　用户定义完整性

用户定义完整性使用户可以定义不属于其他任何完整性类别的特定业务规则。所有完整性类别都支持用户定义完整性，这包括 CREATE TABLE 中所有列级约束和表级约束、存储过程以及触发器。

11.8　小　　结

本章介绍了索引的建立、删除、分析与维护，以及 4 种数据完整性。读者在了解索引概念的前提下，可以使用 SQL Server Management Studio 或者 SQL 语句来建立和删除索引，进而对索引进行分析和维护，以优化对数据的访问。为了保证存储数据的合理性，读者应了解域完整性、实体完整性和引用完整性。

第12章

流程控制

（视频讲解：14 分钟）

本章主要介绍程序的流程控制。通过本章的学习，读者可以掌握基本的流程控制语句来控制程序的执行流程。

学习摘要：

- 分支语句
- 循环控制语句


视频讲解

12.1　流程控制概述

流程控制语句是用来控制程序执行流程的语句。使用流程控制语句可以提高编程语言的处理能力。与程序设计语言（如 C 语言）一样，Transact-SQL 语言提供的流程控制语句如表 12.1 所示。

表 12.1　Transact-SQL 语言提供的流程控制语句

BEGIN...END	CASE	RETURN
IF	WHILE	GOTO
IF...ELSE	WHILE…CONTINUE…BREAK	WAITFOR


视频讲解

12.2　流程控制语句

12.2.1　BEGIN...END

BEGIN...END 语句用于将多个 Transact-SQL 语句组合为一个逻辑块。当流程控制语句必须执行一个包含两条或两条以上的 Transact-SQL 语句的语句块时，使用 BEGIN...END 语句。语法格式如下：

```
BEGIN
{sql_statement...}
END
```

其中，sql_statement 是指包含的 Transact-SQL 语句。

BEGIN 和 END 语句必须成对使用，任何一条语句均不能单独使用。BEGIN 语句后为 Transact-SQL 语句块。最后，END 语句行指示语句块结束。

【例 12.01】　在 BEGIN…END 语句块中完成把两个变量的值交换。在查询编辑器窗口运行的结果如图 12.1 所示。（**实例位置：资源包\源码\12\12.01**）

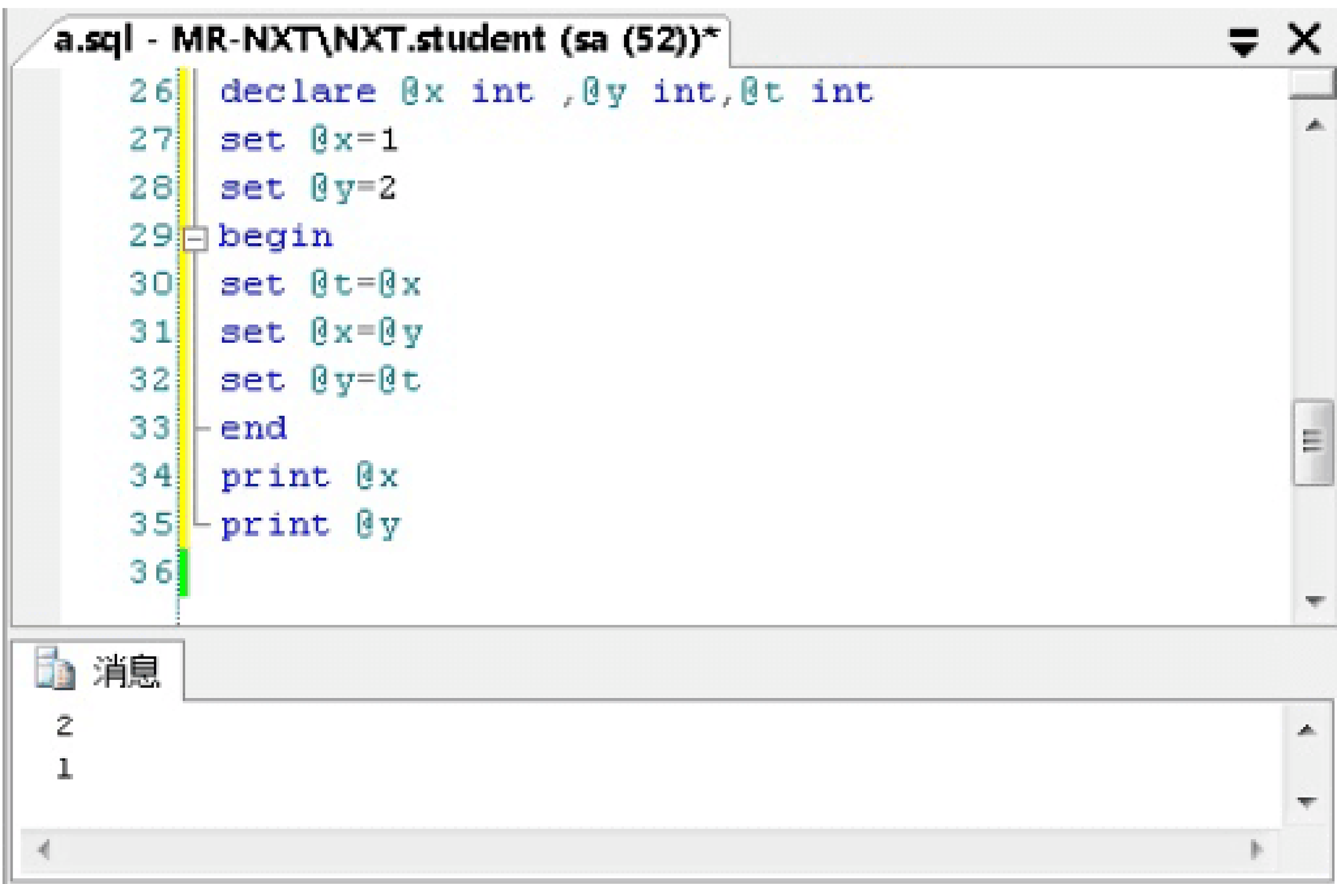


图 12.1　交换两个变量的值

SQL 语句如下：

```
declare @x int, @y int,@t int
set @x=1
set @y=2
begin
set @t=@x
set @x=@y
set @y=@t
end
print @x
print @y
```

此例子不用 BEGIN...END 语句结果也完全一样，但 BEGIN...END 和一些流程控制语句结合起来就有作用了。在 BEGIN...END 中可嵌套另外的 BEGIN...END 来定义另一程序块。

12.2.2　IF

在 SQL Server 中为了控制程序的执行方向，也会像其他语言（如 C 语言）有顺序、选择和循环 3 种控制语句，其中 IF 就属于选择判断结构。IF 结构的语法格式如下：

```
IF<条件表达式>
    {命令行|程序块}
```

其中，<条件表达式>可以是各种表达式的组合，但表达式的值必须是逻辑值“真”或“假”。其中命令行和程序块可以是合法 Transact-SQL 任意语句，但含两条或两条以上的语句的程序块必须加 BEGIN...END 子句。

执行顺序是：遇到选择结构 IF 子句，先判断 IF 子句后的条件表达式，如果条件表达式的逻辑值是“真”，就执行后面的命令行或程序块，然后再执行 IF 结构下一条语句；如果条件式的逻辑值是“假”，就不执行后面的命令行或程序块，直接执行 IF 结构的下一条语句。

【例 12.02】　判断一个数是否是正数。在查询编辑器窗口中运行的结果如图 12.2 所示。（**实例位置：资源包\源码\12\12.02**）

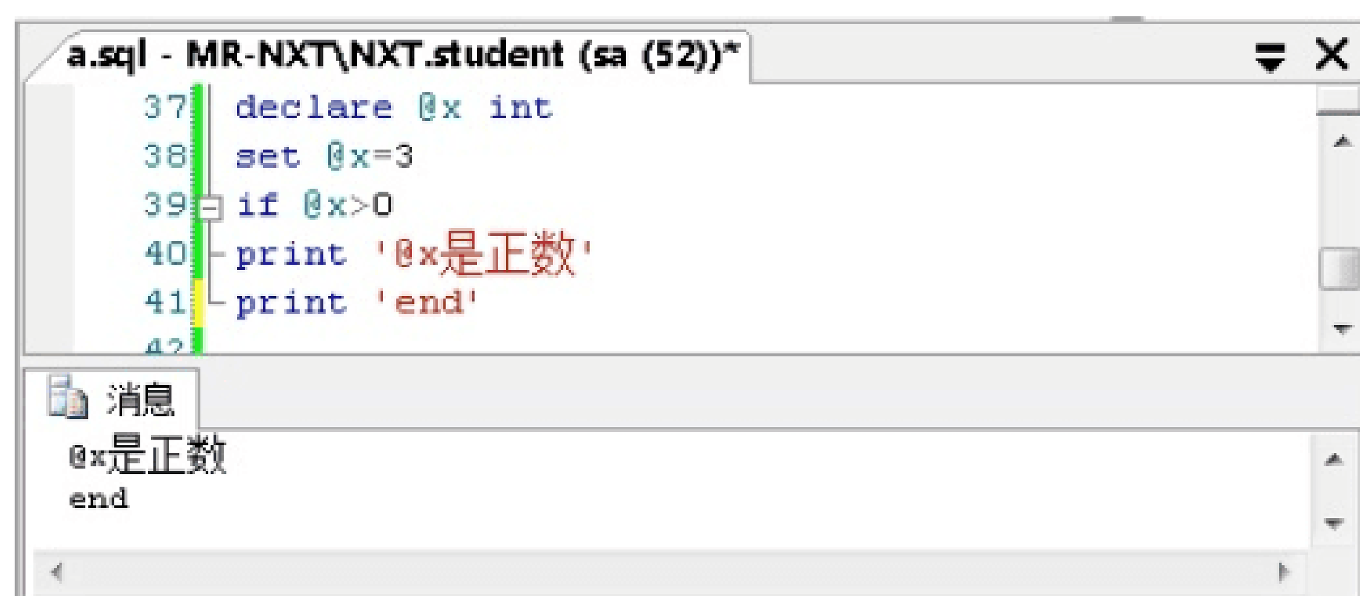


图 12.2　判断一个数的正负

SQL 语句如下：

```
declare @x int
set @x=3
```

```
if @x>0
print '@x 是正数'
print 'end'
```

【例 12.03】 判断一个数的奇偶性。在查询编辑器窗口中运行的结果如图 12.3 所示。（**实例位置：资源包\源码\12\12.03**）

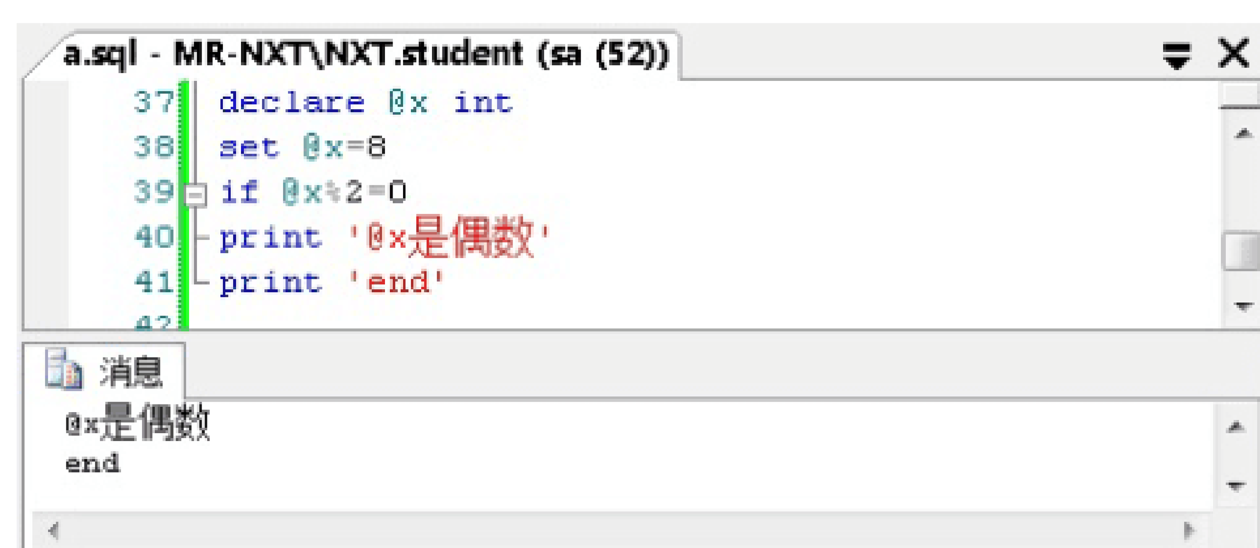


图 12.3 判断一个数的奇偶性

SQL 语句如下：

```
declare @x int
set @x=8
if @x % 2=0
print '@x 偶数'
print 'end'
```

12.2.3 IF…ELSE

IF 选择结构可以带 ElSE 子句。IF...ELSE 的语法格式如下：

```
IF<条件表达式>
    {命令行 1|程序块 1}
[ELSE
    {命令行 2|程序块 2}
```

如果逻辑判断表达式返回的结果是“真”，那么程序接下来会执行命令行 1 或程序块 1；如果逻辑判断表达式返回的结果是“假”，那么程序接下来会执行命令行 2 或程序块 2。无论哪种情况，最后都要执行 IF...ELSE 语句的下一条语句。

【例 12.04】 判断两个数的大小。在查询编辑器窗口运行的结果如图 12.4 所示。（**实例位置：资源包\源码\12\12.04**）

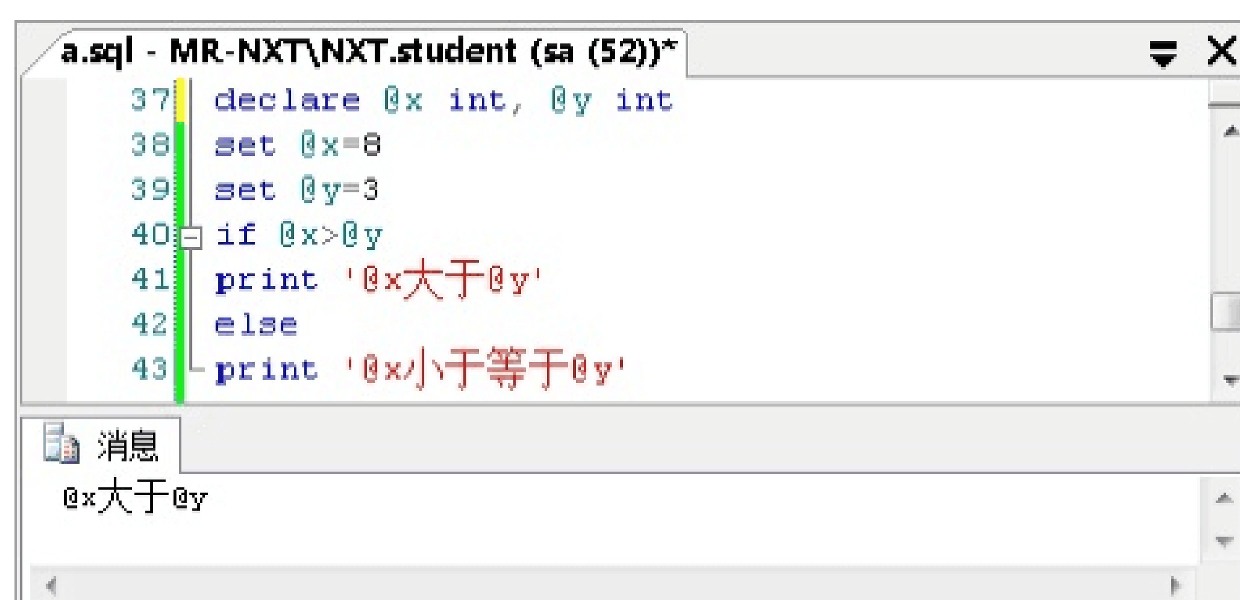


图 12.4 判断两个数的大小

SQL 语句如下：

```
declare @x int,@y int
set @x=8
set @y=3
if @x>@y
print '@x 大于@y'
else
print '@x 小于等于@y'
```

IF...ELSE 结构还可以嵌套解决一些复杂的判断。

【例 12.05】　输入一个坐标值，然后判断它在哪一个象限。在查询编辑器窗口中的运行结果如图 12.5 所示。（**实例位置：资源包\源码\12\12.05**）

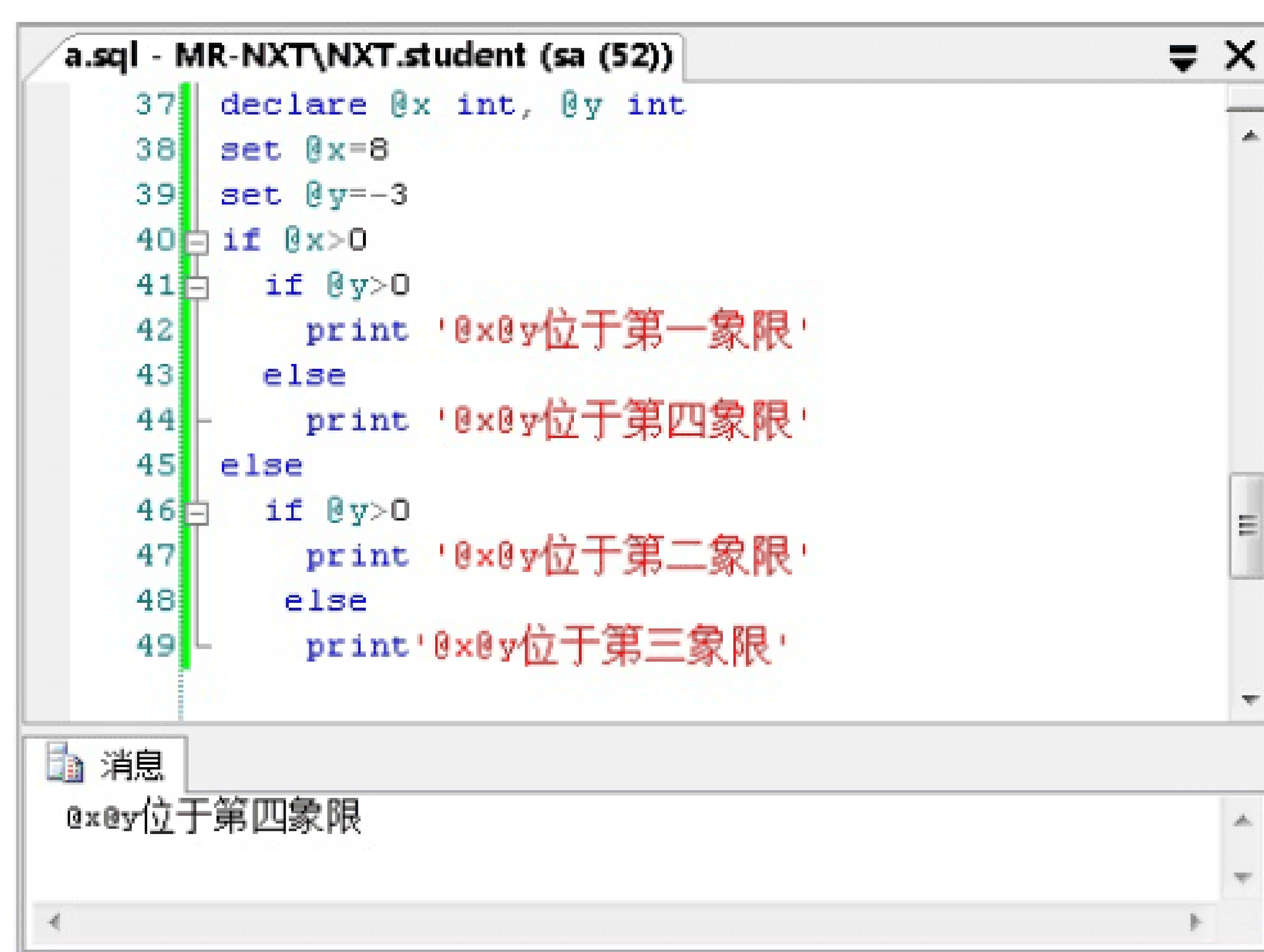


图 12.5　判断坐标位于的象限

SQL 语句如下：

```
declare @x int,@y int
set @x=8
set @y=-3
if @x>0
  if @y>0
    print '@x@y 位于第一象限'
  else
    print '@x@y 位于第四象限'
else
  if @y>0
    print '@x@y 位于第二象限'
  else
    print '@x@y 位于第三象限'
```

12.2.4　CASE

使用 CASE 语句可以很方便地实现多重选择的情况，比 IF...THEN 结构有更多的选择和判断的机

会，可以避免编写多重的 IF...THEN 嵌套循环。

Transact-SQL 支持 CASE 有两种语句格式。

（1）简单 CASE 函数

```
CASE input_expression
  WHEN when_expression THEN result_expression [...n]
  [ELSE else_result_expression]
END
```

（2）CASE 搜索函数

```
CASE
  WHEN Boolean_expression THEN result_expression [...n]
  [ELSE else_result_expression]
END
```

参数说明如下。

- ☑ input_expression：使用简单 CASE 格式时所计算的表达式。input_expression 是任何有效的 Microsoft SQL Server 表达式。
- ☑ WHEN when_expression：使用简单 CASE 格式时 input_expression 所比较的简单表达式。when_expression 是任意有效的 SQL Server 表达式。input_expression 和每个 when_expression 的数据类型必须相同，或者是隐性转换。
- ☑ n：占位符，表明可以使用多个 WHEN when_expression THEN result_expression 子句或 WHEN Boolean_expression THEN result_expression 子句。
- ☑ THEN result_expression：当 input_expression = when_expression 取值为 TRUE，或者 Boolean_expression 取值为 TRUE 时返回的表达式。result_expression 是任意有效的 SQL Server 表达式。
- ☑ ELSE else_result_expression：当比较运算取值不为 TRUE 时返回的表达式。如果省略此参数并且比较运算取值不为 TRUE，CASE 将返回 NULL 值。else_result_expression 是任意有效的 SQL Server 表达式。else_result_expression 和所有 result_expression 的数据类型必须相同，或者必须是隐性转换。
- ☑ WHEN Boolean_expression：使用 CASE 搜索格式时所计算的布尔表达式。Boolean_expression 是任意有效的布尔表达式。

两种格式的执行顺序如下。

（1）简单 CASE 函数执行顺序

- ☑ 计算 input_expression，然后按指定顺序对每个 WHEN 子句的 input_expression = when_expression 进行计算。
- ☑ 返回第一个取值为 TRUE 的 input_expression = when_expression 的 result_expression。
- ☑ 如果没有取值为 TRUE 的 input_expression = when_expression，则当指定 ELSE 子句时，SQL Server 将返回 else_result_expression；若没有指定 ELSE 子句，则返回 NULL 值。

（2）CASE 搜索函数执行顺序

- ☑ 按指定顺序为每个 WHEN 子句的 Boolean_expression 求值。
- ☑ 返回第一个取值为 TRUE 的 Boolean_expression 的 result_expression。

☑　如果没有取值为 TRUE 的 Boolean_expression，则当指定 ELSE 子句时，SQL Server 将返回 else_result_expression；若没有指定 ELSE 子句，则返回 NULL 值。

【例 12.06】　在 pubs 数据库的 titles 表中，使用带有简单 CASE 函数的 SELECT 语句。在查询编辑器窗口中运行的结果如图 12.6 所示。（**实例位置：资源包\源码\12\12.06**）

图 12.6　统计 titles 表

SQL 语句如下：

```
USE pubs
SELECT kind =
        CASE type
            WHEN 'popular_comp' THEN 'Popular Computing'
            WHEN 'mod_cook' THEN 'Modern Cooking'
            WHEN 'business' THEN 'Business'
            WHEN 'psychology' THEN 'Psychology'
            WHEN 'trad_cook' THEN 'Traditional Cooking'
            ELSE 'Not yet categorized'
        END,
    CAST(title AS varchar(30)) AS 'Shortened Title',
    price AS Price
FROM titles
WHERE price IS NOT NULL
ORDER BY type, price
COMPUTE AVG(price) BY type
```

下面的例子应用了第二种 CASE 格式。

【例 12.07】　在 pubs 数据库的 titles 表中，应用第二种 CASE 格式进行查询。在查询编辑器窗口中运行的结果如图 12.7 所示。（**实例位置：资源包\源码\12\12.07**）

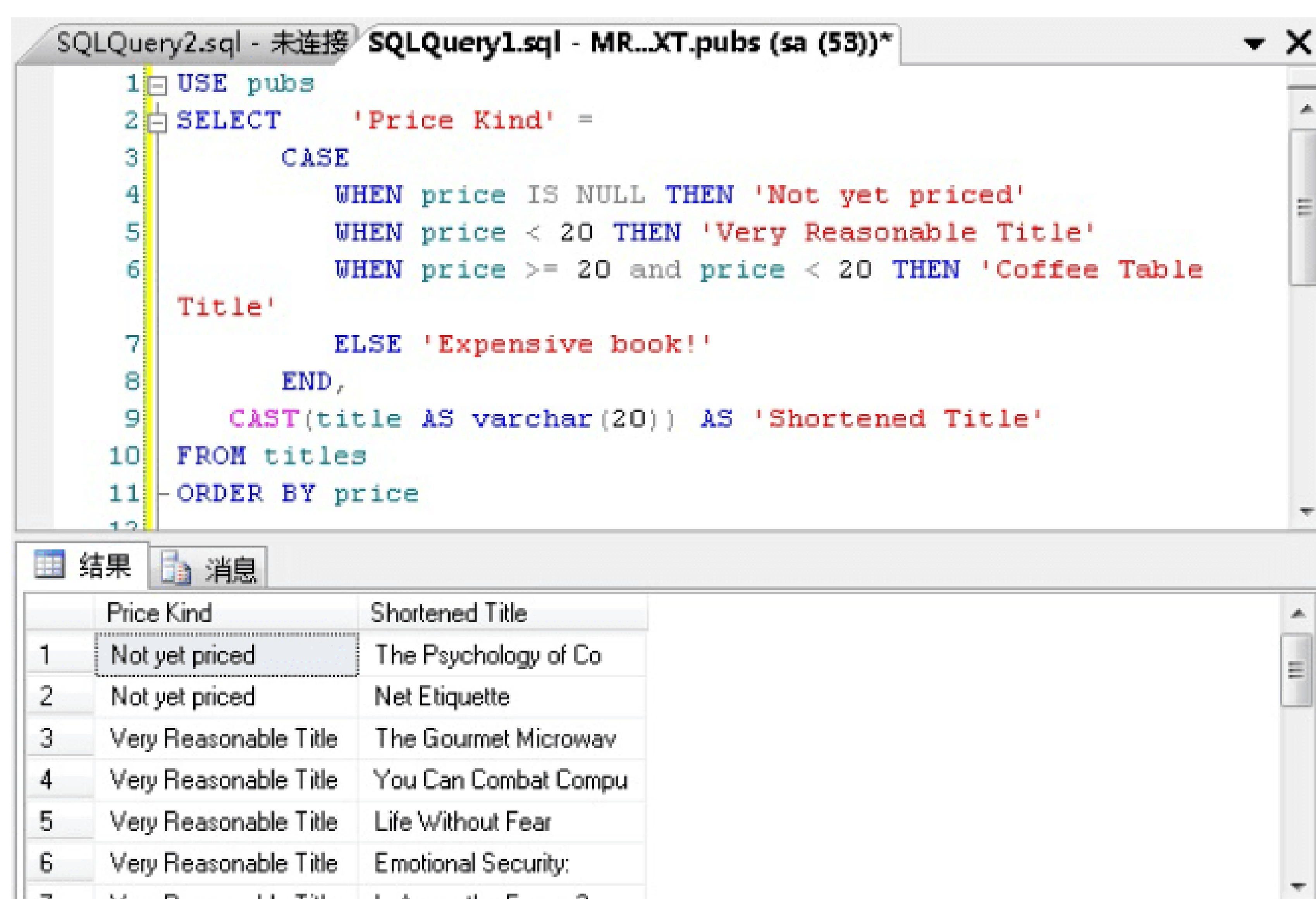


图 12.7　应用第二种格式的 CASE 语句

SQL 语句如下：

```
USE pubs
SELECT 'Price Kind' =
        CASE
            WHEN price IS NULL THEN 'Not yet priced'
            WHEN price < 20 THEN 'Very Reasonable Title'
            WHEN price >= 20 and price < 20 THEN 'Coffee Table Title'
            ELSE 'Expensive book!'
        END,
    CAST(title AS varchar(20)) AS 'Shortened Title'
FROM titles
ORDER BY price
```

12.2.5　WHILE

WHILE 子句是 Transact-SQL 语句支持的循环结构。在条件为“真”的情况下，WHILE 子句可以循环地执行其后的一条 Transact-SQL 命令。如果想循环执行一组命令，则需要使用 BEGIN…END 子句。语法格式如下：

```
WHILE<条件表达式>
BEGIN
     <命令行|程序块>
END
```

遇到 WHILE 子句，先判断条件表达式的值，当条件表达式的值为“真”时，执行循环体中的命令行或程序块，遇到 END 子句会自动地再次判断条件表达式值的真假，决定是否执行循环体中的语句。只能当条件表达式的值为“假”时，才结束执行循环体的语句。

【例 12.08】　求数字 1～10 的和。在查询编辑器窗口中运行的结果如图 12.8 所示。（**实例位置：**

资源包\源码\12\12.08）

SQL 语句如下：

```
declare @n int,@sum int
set @n=1
set @sum=0
while @n<=10
begin
set @sum=@sum+@n
set @n=@n+1
end
print @sum
```

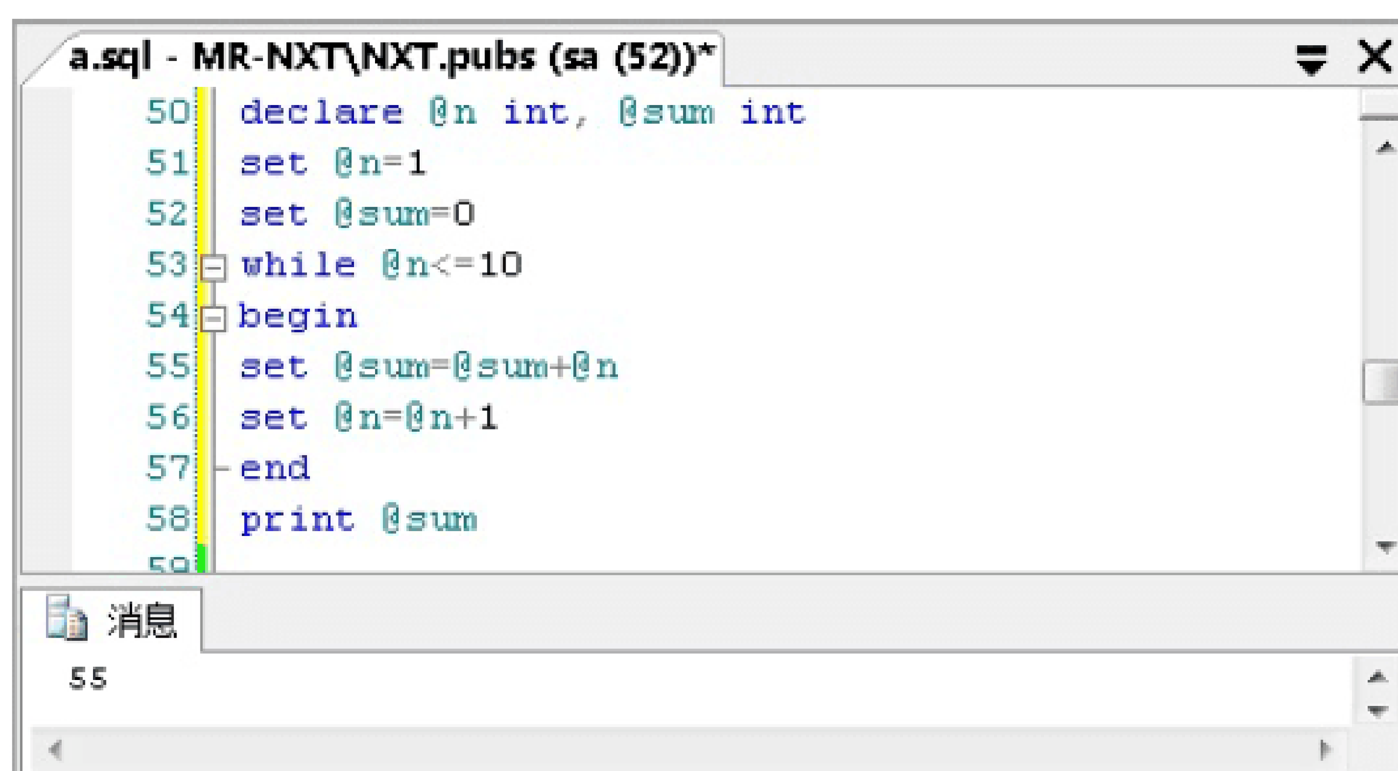


图 12.8　求数字 1～10 的和

12.2.6　WHILE…CONTINUE…BREAK

循环结构 WHILE 子句还可以用 CONTINUE 和 BREAK 命令控制 WHILE 循环中语句的执行。语法格式如下：

```
WHILE<条件表达式>
BEGIN
    <命令行|程序块>
    [BREAK]
    [CONTINUE]
    [命令行|程序块]
END
```

其中，CONTINUTE 命令可以让程序跳过 CONTINUE 命令之后的语句，回到 WHILE 循环的第一行命令。BREAK 命令则让程序完全跳出循环，结束 WHILE 命令的执行。

【例 12.09】　求 1～10 之间的偶数的和，并用 CONTINUE 控制语句的输出。在查询编辑器窗口中运行的结果如图 12.9 所示。（**实例位置：资源包\源码\12\12.09**）

SQL 语句如下：

```
declare @x int,@sum int
set @x=1
```

```
set @sum=0
while @x<10
begin
set @x=@x+1
if @x%2=0
set @sum=@sum+@x
else
continue
print '只有@x 是偶数才输出这句话'
end
print @sum
```



图 12.9　求 1～10 之间偶数的和

12.2.7　RETURN

RETURN 语句用于从查询过程中无条件退出。RETURN 语句可在任何时候用于从过程、批处理或语句块中退出。位于 RETURN 之后的语句不会被执行。语法格式如下：

```
RETURN[整数值]
```

在括号内可指定一个返回值。如果没有指定返回值，SQL Server 系统会根据程序执行的结果返回一个内定值，内定值如表 12.2 所示。

表 12.2　RETURN 命令返回的内定值

返　回　值	含　　义
0	程序执行成功
−1	找不到对象
−2	数据类型错误
−3	死锁
−4	违反权限原则

续表

返　回　值	含　　义
−5	语法错误
−6	用户造成的一般错误
−7	资源错误，如磁盘空间不足
−8	非致命的内部错误
−9	已达到系统的极限
−10 或−11	致命的内部不一致性错误
−12	表或指针破坏
−13	数据库破坏
−14	硬件错误

【例 12.10】　RETURN 语句的使用。在查询编辑器窗口中运行的结果如图 12.10 所示。（**实例位置：资源包\源码\12\12.10**）

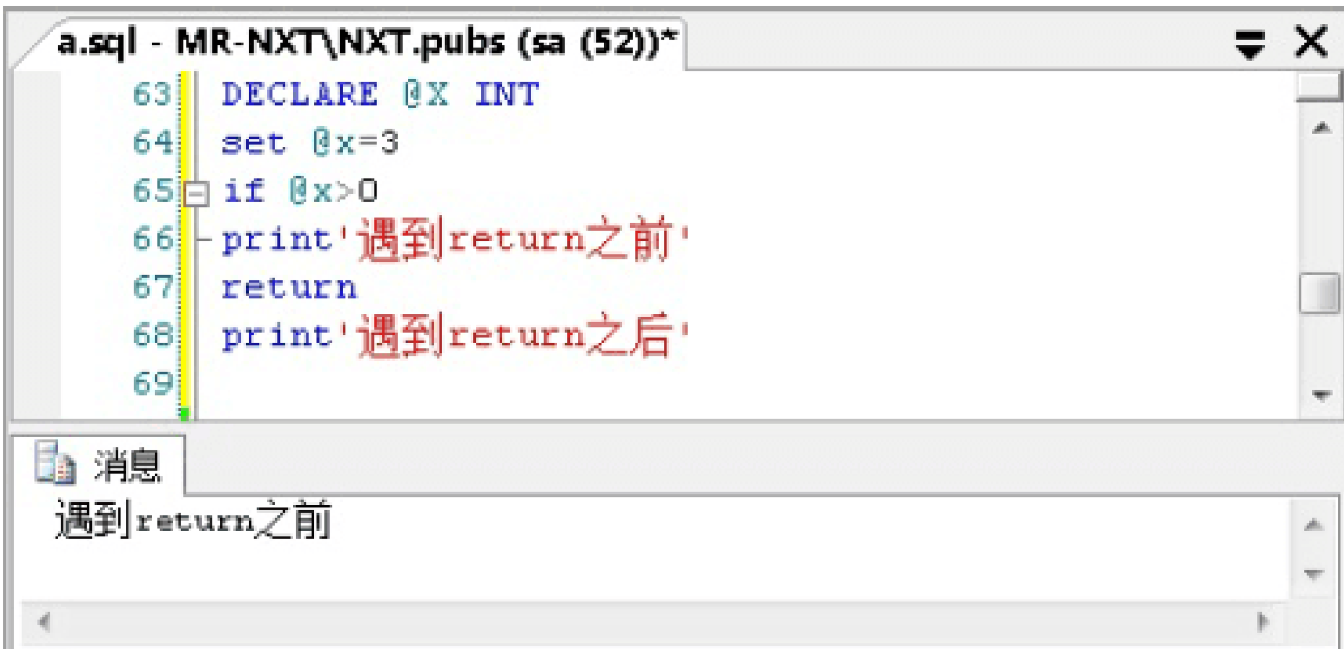


图 12.10　RETURN 语句的应用

SQL 语句如下：

```
DECLARE @X INT
set @x=3
if @x>0
print'遇到 return 之前'
return
print'遇到 return 之后'
```

12.2.8　GOTO

GOTO 命令用来改变程序执行的流程，使程序跳到标识符指定的程序行再继续往下执行。语法格式如下：

```
GOTO　标识符
```

标识符需要在其名称后加上一个冒号“:”。例如，“33:”“loving:”。

【例 12.11】　用 GOTO 语句实现跳转输入其下的值。在查询编辑器窗口中执行的结果如图 12.11 所示。（**实例位置：资源包\源码\12\12.11**）

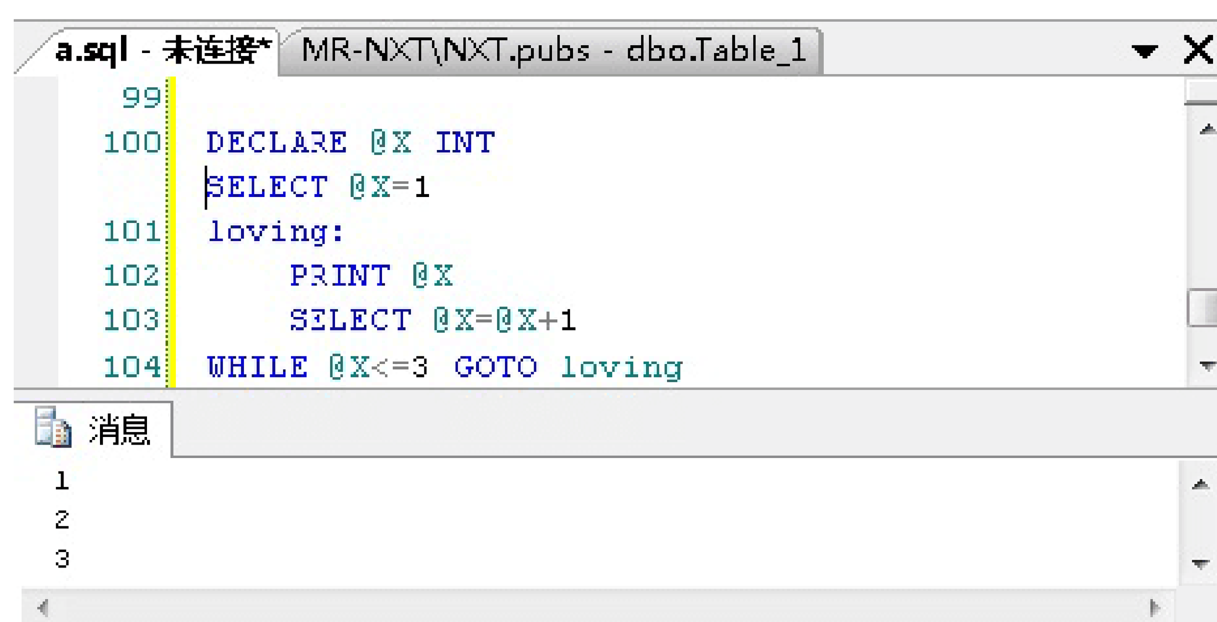


图 12.11　GOTO 语句的应用

SQL 语句如下：

```
DECLARE @X INT
SELECT @X=1
loving:
    PRINT @X
    SELECT @X=@X+1
WHILE @X<=3 GOTO loving
```

12.2.9　WAITFOR

WAITFOR 指定触发器、存储过程或事务执行的时间、时间间隔或事件；还可以用来暂时停止程序的执行，直到所设定的等待时间已过才继续往下执行。语法格式如下：

```
WAITFOR{DELAY<'时间'>|TIME<'时间'>}
```

其中“时间”必须为 DATETIME 类型的数据，如 11:15:27，但不能包括日期。各关键字含义如下。

☑　DELAY：用来设定等待的时间，最多可达 24 小时。

☑　TIME：用来设定等待结束的时间点。

例如，再过 3 秒钟显示“葱葱睡觉了！”，SQL 语句如下：

```
WAITFOR DELAY '00:00:03'
PRINT '葱葱睡觉了！ '
```

例如，等到 15 点显示“喜爱的歌曲：舞”，SQL 语句如下：

```
WAITFOR TIME '15:00:00'
PRINT '喜爱的歌曲：舞'
```

12.3　小　　结

本章介绍了常用的流程控制语句，流程控制语句能够控制程序的执行顺序，其中条件判断语句和循环控制语句十分重要。条件判断语句包括 IF、IF…ELSE 和 CASE。循环控制语句包括 WHILE、WHILE…CONTINUE…BREAK。跳转语句包括 RETURN 和 GOTO。

第13章

存储过程

（视频讲解：20分钟）

存储过程（Stored Procedure）代替了传统的逐条执行SQL语句的方式。存储过程是预编译SQL语句的集合，这些语句存储在一个名称下并作为一个单元来处理。一个存储过程中可包含查询、插入、删除、更新等操作的一系列SQL语句，当这个存储过程被调用执行时，这些操作也会同时执行。

学习摘要：

- **存储过程的基本概念**
- **创建存储过程的两种方法**
- **执行存储过程**
- **使用 sys.sql_modules 查看存储过程的定义**
- **使用 ALTER PROCEDURE 语句修改存储过程**
- **存储过程重命名**
- **删除存储过程**



13.1 存储过程概述

13.1.1 存储过程的概念

存储过程（Stored Procedure）是预编译 SQL 语句的集合，这些语句存储在一个名称下并作为一个单元来处理。存储过程代替了传统的逐条执行 SQL 语句的方式。一个存储过程中可包含查询、插入、删除、更新等操作的一系列 SQL 语句，当这个存储过程被调用执行时，这些操作也会同时执行。

存储过程与其他编程语言中的过程类似，它可以接收输入参数并以输出参数的格式向调用过程或批处理返回多个值；包含用于在数据库中执行操作（包括调用其他过程）的编程语句；向调用过程或批处理返回状态值，以指明成功或失败（以及失败的原因）。

SQL Server 提供了 3 种类型的存储过程，各类型存储过程如下。

☑ 系统存储过程：用来管理 SQL Server 和显示有关数据库和用户的信息的存储过程。

☑ 自定义存储过程：用户在 SQL Server 中通过采用 SQL 语句创建存储过程。

☑ 扩展存储过程：通过编程语言（如 C 语言）创建外部例程，并将这个例程在 SQL Server 中作为存储过程使用。

13.1.2 存储过程的优点

存储过程的优点表现在以下几个方面。

（1）存储过程可以嵌套使用，支持代码重用。

（2）存储过程可以接收与使用参数动态执行其中的 SQL 语句。

（3）存储过程比一般的 SQL 语句执行速度快。存储过程在创建时已经被编译，每次执行时不需要重新编译。而 SQL 语句每次执行都需要编译。

（4）存储过程具有安全特性（如权限）和所有权链接，以及可以附加到它们的证书。用户可以被授予权限来执行存储过程而不必直接对存储过程中引用的对象具有权限。

（5）存储过程允许模块化程序设计。存储过程一旦创建，以后即可在程序中调用任意多次。这可以改进应用程序的可维护性，并允许应用程序统一访问数据库。

（6）存储过程可以减少网络通信流量。一个需要数百行 SQL 语句代码的操作可以通过一条执行过程代码的语句来执行，而不需要在网络中发送数百行代码。

（7）存储过程可以强制应用程序的安全性。参数化存储过程有助于保护应用程序不受 SQL Injection（SQL 注入）攻击。


说明

SQL Injection 是一种攻击方法，它可以将恶意代码插入到以后将传递给 SQL Server 供分析和执行的字符串中。任何构成 SQL 语句的过程都应进行注入漏洞检查，因为 SQL Server 将执行其接收到的所有语法有效的查询。



13.2　创建存储过程

存储过程（Stored Procedure）是在数据库服务器端执行的一组 Transact-SQL 语句的集合，经编译后存放在数据库服务器中。本节主要介绍如何通过 SQL Server Management Studio 和 Transact-SQL 语句创建存储过程。

13.2.1　使用向导创建存储过程

在 SQL Server 2014 中，使用向导创建存储过程的步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）在“对象资源管理器”中选择指定的服务器和数据库，展开数据库的“可编辑性”节点，鼠标右键单击“存储过程”，在弹出的快捷菜单中选择“新建存储过程”命令，如图 13.1 所示。

（3）在弹出的“连接到数据库引擎”窗口中，单击“连接”按钮，便出现创建存储过程窗口，如图 13.2 所示。

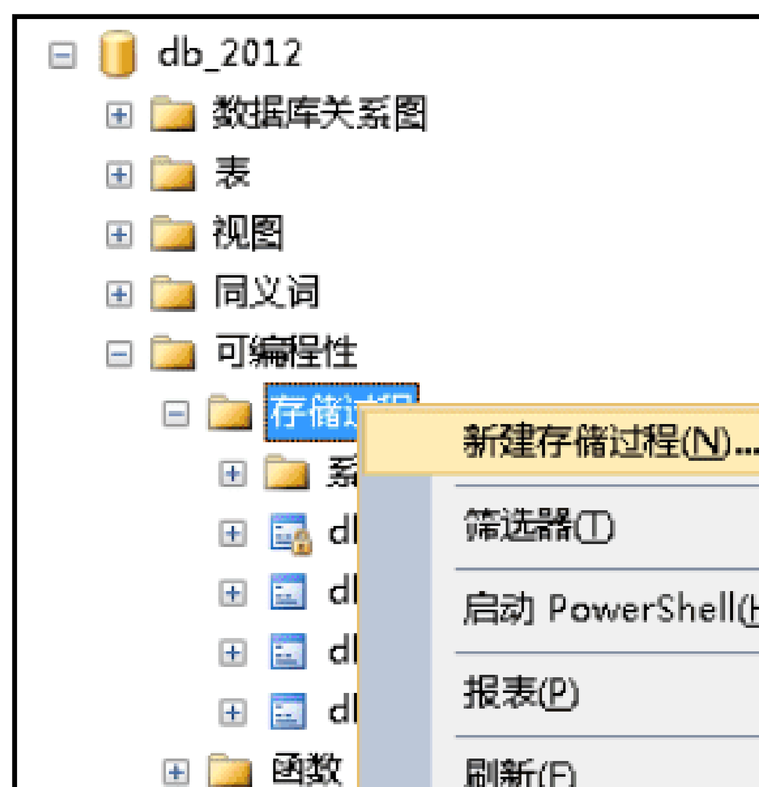


图 13.1　选择“新建存储过程”命令

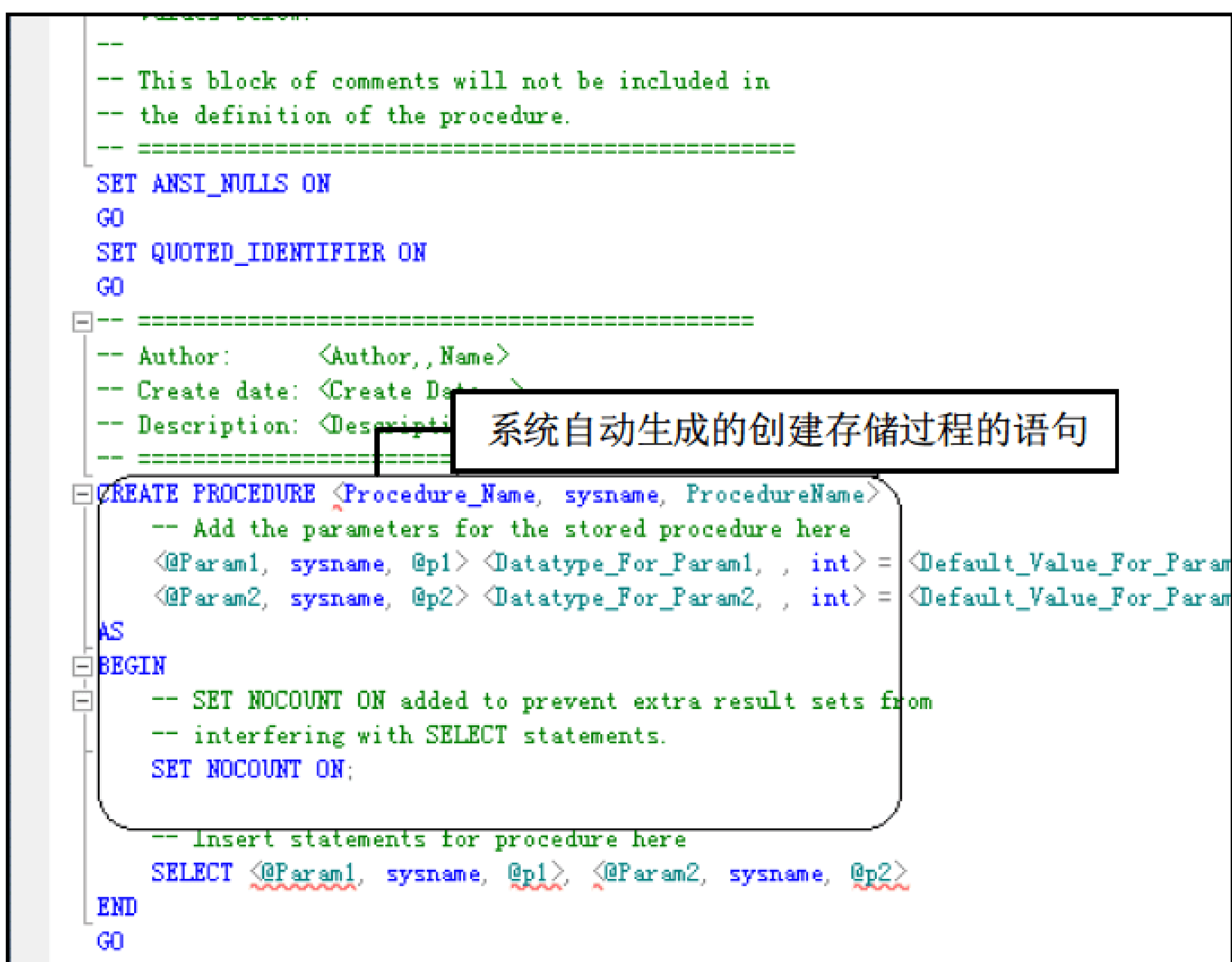


图 13.2　创建存储过程窗口

在创建存储过程窗口的文本框中，可以看到系统自动给出了创建存储过程的格式模板语句，可以在此模板中进行修改来创建新的存储过程。

【例 13.01】　创建一个名称为 Proc_Stu 的存储过程，要求完成以下功能：在 Student 表中查询男生的 Sno、Sname、Sex、Sage 这几个字段的内容。（**实例位置：资源包\源码\13\13.01**）

具体的操作步骤如下。

（1）在创建存储过程窗口中单击“查询”菜单，选择“指定模板参数的值”，弹出“指定模板参

数的值”对话框，如图 13.3 所示。

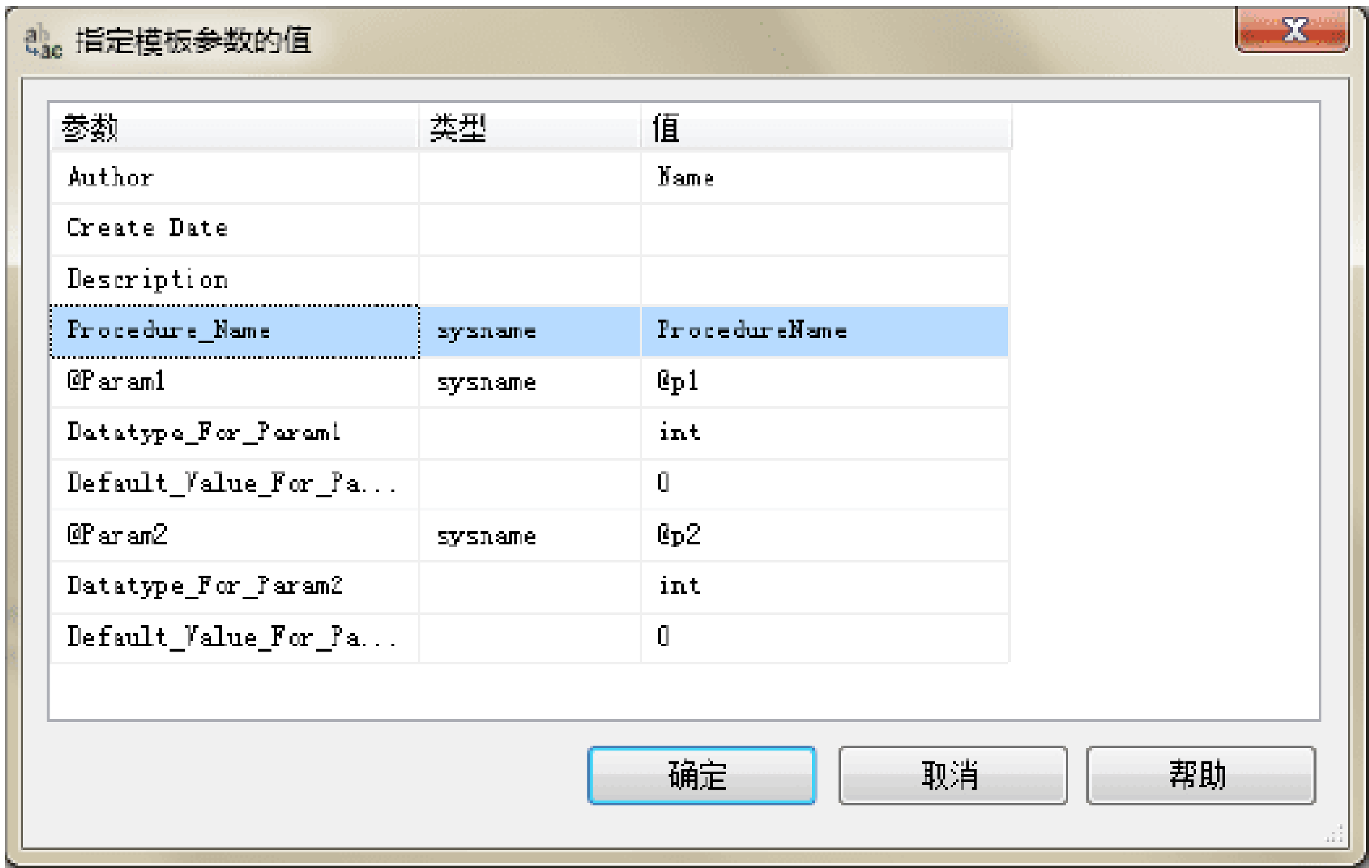


图 13.3　指定模板参数的值

（2）在“指定模板参数的值”对话框中将 Procedure_Name 参数对应的名称修改为 Proc_Stu，单击“确定”按钮，关闭此对话框。

（3）在创建存储过程窗口中，将对应的 SELECT 语句修改为以下语句：

```
SELECT Sno,Sname,Sex,Sage
FROM Student
WHERE Sex='男'
```

13.2.2　使用 CREATE PROC 语句创建存储过程

在 SQL 中，可以使用 CREATE PROCEDURE 语句创建存储过程，其语法格式如下：

```
CREATE PROC [EDURE] procedure_name [; number]
    [{@parameter data_type}
        [VARYING] [= default] [OUTPUT]
   ] [,...n]
AS sql_statement
```

CREATE PROC 语句的参数及说明如表 13.1 所示。

表 13.1　CREATE PROC 语句的参数及说明

参　　数	描　　述
CREATE PROCEDURE	关键字，也可以写成 CREATE PROC
procedure_name	创建的存储过程名称
number	对存储过程进行分组
@parameter	存储过程参数，存储过程可以声明一个或多个参数
data_type	参数的数据类型，所有数据类型（包括 text、ntext 和 image）均可以用作存储过程的参数，但是 cursor 数据类型只能用于 OUTPUT 参数
VARYING	可选项，指定作为输出参数支持的结果集（由存储过程动态构造，内容可以变化），该关键字仅适用于游标参数

续表

参　　数	描　　述
default	可选项，表示为参数设置默认值
OUTPUT	可选项，表明参数是返回参数，可以将参数值返回给调用的过程
n	表示可以定义多个参数
AS	指定存储过程要执行的操作
sql_statement	存储过程中的过程体

【例 13.02】　使用 CREATE PROCEDURE 语句创建一个存储过程，用来根据学生编号查询学生信息。(**实例位置：资源包\源码\13\13.02**)

SQL 语句如下：

```
Create Procedure Proc_Student
@Proc_Sno int
as
select * from Student where Sno = @Proc_Sno
```

查询结果如图 13.4 所示。

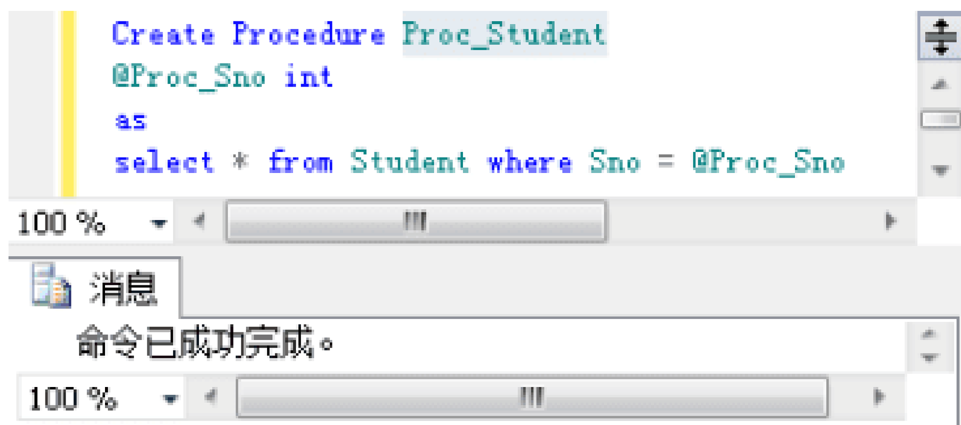


图 13.4　创建存储过程

13.3　管理存储过程


视频讲解

存储过程创建完成后，用户可以通过 SQL Server Management Studio 工具对其进行管理。数据库中的存储过程都被保存在“数据库”→“数据库名称”→“可编程性”→“存储过程”路径下。本节介绍使用 SQL Server Management Studio 工具对存储过程进行执行、查看代码、修改代码及名称、删除等管理。

13.3.1　执行存储过程

存储过程创建完成后，可以通过 EXECUTE 执行，可简写为 EXEC。

1. EXECUTE

EXECUTE 用来执行 Transact-SQL 中的命令字符串、字符串或执行下列模块之一：系统存储过程、用户定义存储过程、标量值用户定义函数或扩展存储过程。

EXECUTE 的语法格式如下：

```
[{EXEC | EXECUTE}]
    {
        [@return_status =]
        {module_name [;number] | @module_name_var}
        [[@parameter =] {value
                            | @variable [OUTPUT]
                            | [DEFAULT]
                            }
         ]
        [,...n]
        [WITH RECOMPILE]
    }
[;]
```

EXECUTE 语句的参数及说明如表 13.2 所示。

表 13.2　EXECUTE 语句的参数及说明

参　　数	描　　述
@return_status	可选的整型变量，存储模块的返回状态。这个变量在用于 EXECUTE 语句前，必须在批处理、存储过程或函数中声明过
module_name	是要调用的存储过程或标量值用户定义函数的完全限定或者不完全限定名称。模块名称必须符合标识符规则。无论服务器的排序规则如何，扩展存储过程的名称总是区分大小写
number	是可选整数，用于对同名的过程分组。该参数不能用于扩展存储过程
@module_name_var	是局部定义的变量名，代表模块名称
@parameter	module_name 的参数，与在模块中定义的相同。参数名称前必须加上 at 符号（@）
value	传递给模块或传递命令的参数值。如果参数名称没有指定，参数值必须以在模块中定义的顺序提供
@variable	是用来存储参数或返回参数的变量
OUTPUT	指定模块或命令字符串返回一个参数。该模块或命令字符串中的匹配参数也必须已使用关键字 OUTPUT 创建。使用游标变量作为参数时使用该关键字
DEFAULT	根据模块的定义，提供参数的默认值。当模块需要的参数值没有定义默认值并且缺少参数或指定了 DEFAULT 关键字，会出现错误
WITH RECOMPILE	执行模块后，强制编译、使用和放弃新计划。如果该模块存在现有查询计划，则该计划将保留在缓存中

2．使用 EXECUTE 执行存储过程

【例 13.03】　使用 EXECUTE 执行存储过程 Proc_Stu。（**实例位置：资源包\源码\13\13.03**）

SQL 语句如下：

```
exec Proc_Stu
```

使用 EXECUTE 执行存储过程的步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）单击工具栏中的新建查询(N)按钮，新建查询编辑器，并输入如下 SQL 语句代码。

```
exec Proc_Stu
```

（3）单击!执行(X)按钮，就可以执行上述 SQL 语句代码，即可完成执行 Proc_Stu 存储过程。执行结果如图 13.5 所示。

图 13.5　执行存储过程的结果

13.3.2　查看存储过程

许多系统存储过程、系统函数和目录视图都提供有关存储过程的信息。用户可以使用这些系统存储过程来查看存储过程的定义，即用于创建存储过程的 Transact-SQL 语句。

可以通过下面 3 种系统存储过程和目录视图查看存储过程。

1．使用 sys.sql_modules 查看存储过程的定义

sys.sql_modules 为系统视图，通过该视图可以查看数据库中的存储过程。查看存储过程的操作方法如下。

（1）单击工具栏中的新建查询(N)按钮，新建查询编辑器。

（2）在新建查询编辑器中输入如下代码：

```
SELECT * FROM sys.sql_modules
```

（3）单击!执行(X)按钮，执行该查询命令。查询结果如图 13.6 所示。

图 13.6　使用 sys.sql_modules 视图查询的存储过程

2．使用 OBJECT_DEFINITION 查看存储过程的定义

返回指定对象定义的 Transact-SQL 源文本。语法格式如下：

```
OBJECT_DEFINITION(object_id)
```

其中，object_id 是指要使用的对象的 ID。object_id 的数据类型为 int，并假定表示当前数据库上下文中的对象。

【例 13.04】 使用 OBJECT_DEFINITION 查看 ID 为 309576141 存储过程的代码。（**实例位置：资源包\源码\13\13.04**）

SQL 语句如下：

```
SELECT OBJECT_DEFINITION(309576141)
```

3. 使用 sp_helptext 查看存储过程的定义

显示用户定义规则的定义、默认值、未加密的 Transact-SQL 存储过程、用户定义 Transact-SQL 函数、触发器、计算列、CHECK 约束、视图或系统对象（如系统存储过程）。语法格式如下：

```
sp_helptext [@objname =] 'name' [, [@columnname =] computed_column_name]
```

参数说明如下。

☑ [@objname =] 'name'：架构范围内的用户定义对象的限定名称和非限定名称。仅当指定限定对象时才需要引号。如果提供的是完全限定名称（包括数据库名称），则数据库名称必须是当前数据库的名称。对象必须在当前数据库中。name 的数据类型为 nvarchar(776)，无默认值。

☑ [@columnname =] 'computed_column_name'：要显示其定义信息的计算列的名称。必须将包含列的表指定为 name。column_name 的数据类型为 sysname，无默认值。

【例 13.05】 通过 sp_helptext 系统存储过程查看名为 Proc_Stu 存储过程的代码。（**实例位置：资源包\源码\13\13.05**）

SQL 语句如下：

```
sp_helptext 'Proc_Stu'
```

操作步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）选择存储过程所在的数据库，如 db_2014 数据库。

（3）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，并输入如下 SQL 语句代码。

```
sp_helptext 'Proc_Stu'
```

（4）单击! 执行(X) 按钮，就可以执行上述 SQL 语句代码。执行结果如图 13.7 所示。

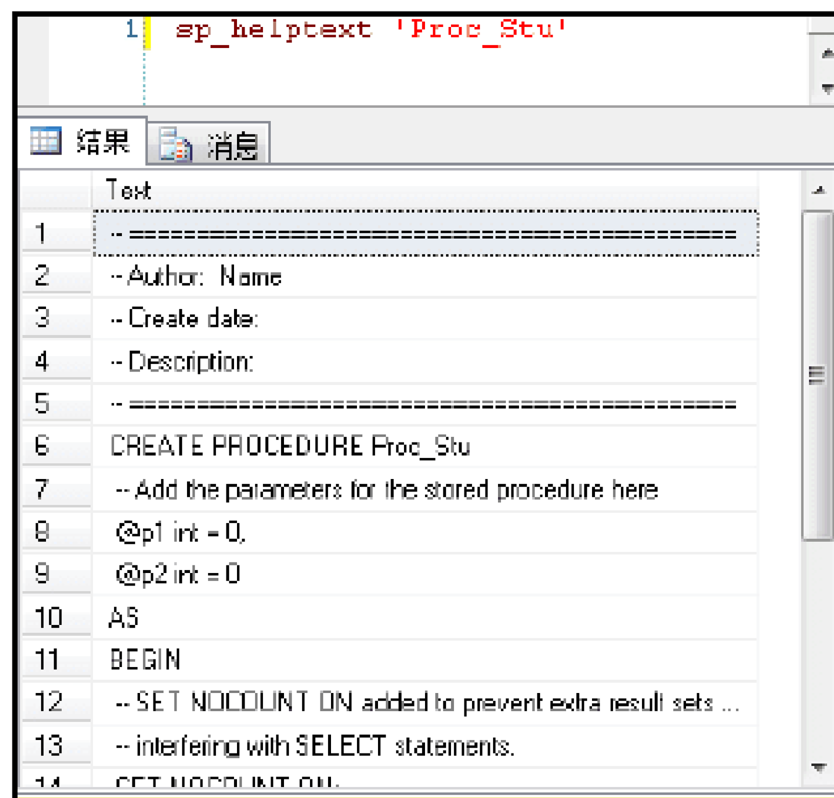


图 13.7　查看 Proc_Stu 存储过程的结果

13.3.3　修改存储过程

修改存储过程可以改变存储过程中的参数或者语句，可以通过 SQL 语句中的 ALTER PROCEDURE 语句实现。虽然删除并重新创建该存储过程，也可以达到修改存储过程的目的，但是将丢失与该存储过程关联的所有权限。

1．ALTER PROCEDURE 语句

ALTER PROCEDURE 语句用来修改通过执行 CREATE PROCEDURE 语句创建的存储过程。该语句修改存储过程时不会更改权限，也不影响相关的存储过程或触发器。

ALTER PROCEDURE 语句的语法格式如下：

```
ALTER {PROC | PROCEDURE} [schema_name.] procedure_name [; number]
    [{@parameter [type_schema_name.] data_type}
    [VARYING] [= default] [OUT PUT]
    ] [,...n]
[WITH <procedure_option> [,...n]]
[FOR REPLICATION]
AS {<sql_statement> [...n] | <method_specifier>}
<procedure_option> ::=
    [ENCRYPTION]
    [RECOMPILE]
    [EXECUTE_AS_Clause]
<sql_statement> ::=
{[BEGIN] statements [END]}
<method_specifier> ::=
EXTERNAL NAME
assembly_name.class_name.method_name
```

ALTER PROCEDURE 语句的参数及说明如表 13.3 所示。

表 13.3　ALTER PROCEDURE 语句的参数及说明

参　数	描　述
schema_name	过程所属架构的名称
procedure_name	要更改的过程的名称。过程名称必须符合标识符规则
number	现有的可选整数，该整数用来对具有同一名称的过程进行分组，以便可以用一个 DROP PROCEDURE 语句全部删除它们
@parameter	过程中的参数。最多可以指定 2100 个参数
[type_schema_name.] data_type	参数及其所属架构的数据类型
VARYING	指定作为输出参数支持的结果集。此参数由存储过程动态构造，并且其内容可以不同。仅适用于游标参数
default	参数的默认值
OUTPUT	指示参数是返回参数

续表

参　数	描　述
FOR REPLICATION	指定不能在订阅服务器上执行为复制创建的存储过程
AS	过程将要执行的操作
ENCRYPTION	指示数据库引擎会将 ALTER PROCEDURE 语句的原始文本转换为模糊格式
RECOMPILE	指示 SQL Server 2014 数据库引擎不会缓存该过程的计划，该过程在运行时重新编译
EXECUTE AS	指定访问存储过程后执行该存储过程所用的安全上下文
<sql_statement>	过程中要包含的任意数目和类型的 Transact-SQL 语句。但有一些限制
EXTERNAL NAME assembly_name.class_name.method_name	指定 Microsoft .NET Framework 程序集的方法，以便 CLR 存储过程引用。class_name 必须为有效的 SQL Server 标识符，并且必须作为类存在于程序集中。如果类具有使用句点（.）分隔命名空间部分的命名空间限定名称，则必须使用方括号（[]）或引号（" "）来分隔类名。指定的方法必须为该类的静态方法



注意

默认情况下，SQL Server 不能执行 CLR 代码。可以创建、修改和删除引用公共语言运行时模块的数据库对象；不过，只有在启用 clr enabled 选项之后，才能在 SQL Server 中执行这些引用。若要启用该选项，请使用 sp_configure。

2．使用 ALTER PROCEDURE 语句修改存储过程

【例 13.06】　通过 ALTER PROCEDURE 语句修改名为 Proc_Stu 的存储过程。（**实例位置：资源包\源码\13\13.06**）

具体操作步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）选择存储过程所在的数据库，如 db_2014 数据库。

（3）单击工具栏中的新建查询(N)按钮，新建查询编辑器，并输入如下 SQL 语句代码。

```
ALTER PROCEDURE [dbo].[Proc_Stu]
@Sno varchar(10)
as
select * from student
```

（4）单击执行(X)按钮，就可以执行上述 SQL 语句代码。执行结果如图 13.8 所示。

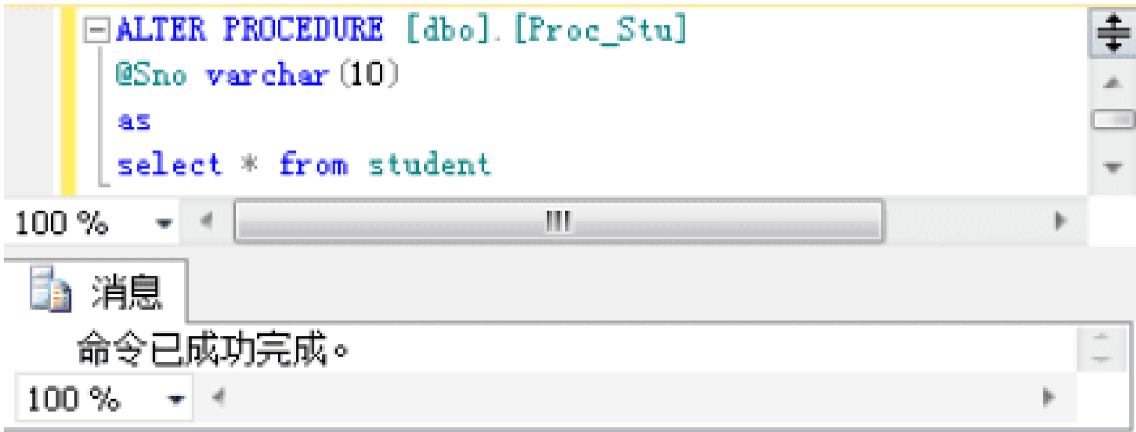


图 13.8　使用 ALTER PROCEDURE 语句修改存储过程

除了上述方法修改存储过程外，也可以通过 SQL Server 2014 自动生成的 ALTER PROCEDURE 语句修改存储过程。以修改系统数据库 master 中系统存储过程 sp_MScleanupmergepublisher_internal 为例，

操作步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）展开“对象资源管理器”中“数据库”→“系统数据库”→master→“可编程性”→“系统存储过程”的节点后，在 sp_MScleanupmergepublisher_internal 系统存储过程上单击鼠标右键，弹出快捷菜单，如图 13.9 所示。



图 13.9　修改存储过程

（3）选择“修改”命令，在查询编辑器窗口中自动生成修改该存储过程的语句。生成的语句如图 13.10 所示。

```
USE [master]
GO
/****** Object:  StoredProcedure [sys].[sp_MScleanupmergepublisher_internal]    Script Date: 2018/7/16 星期一 14:58:31 ******/
SET ANSI_NULLS OFF
GO
SET QUOTED_IDENTIFIER OFF
GO

ALTER procedure [sys].[sp_MScleanupmergepublisher_internal]
as
begin
    set nocount on
    declare @status_mask int
    declare @published_mask int
    declare @published_database_name sysname
    declare @command nvarchar(4000)

    -- Security check: sysadmin only
    if (isnull(is_srvrolemember('sysadmin'),0) = 0)
    begin
        raiserror(14260, 16,-1)
        return (1)
    end
```

图 13.10　自动生成的 SQL 语句

（4）修改该段 SQL 语句并执行，即可完成修改该存储过程。

13.3.4　重命名存储过程

重新命名存储过程可以通过手动操作或执行 sp_rename 系统存储过程实现。

1．手动操作重新命名存储过程

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）展开“对象资源管理器”中“数据库”→“数据库名称”→“可编程性”→“存储过程”节点，鼠标右键单击需要重新命名的存储过程，在弹出的快捷菜单中选择“重命名”命令。例如，修改 db_2014 数据库中的 Proc_stu 存储过程名称，如图 13.11 所示。

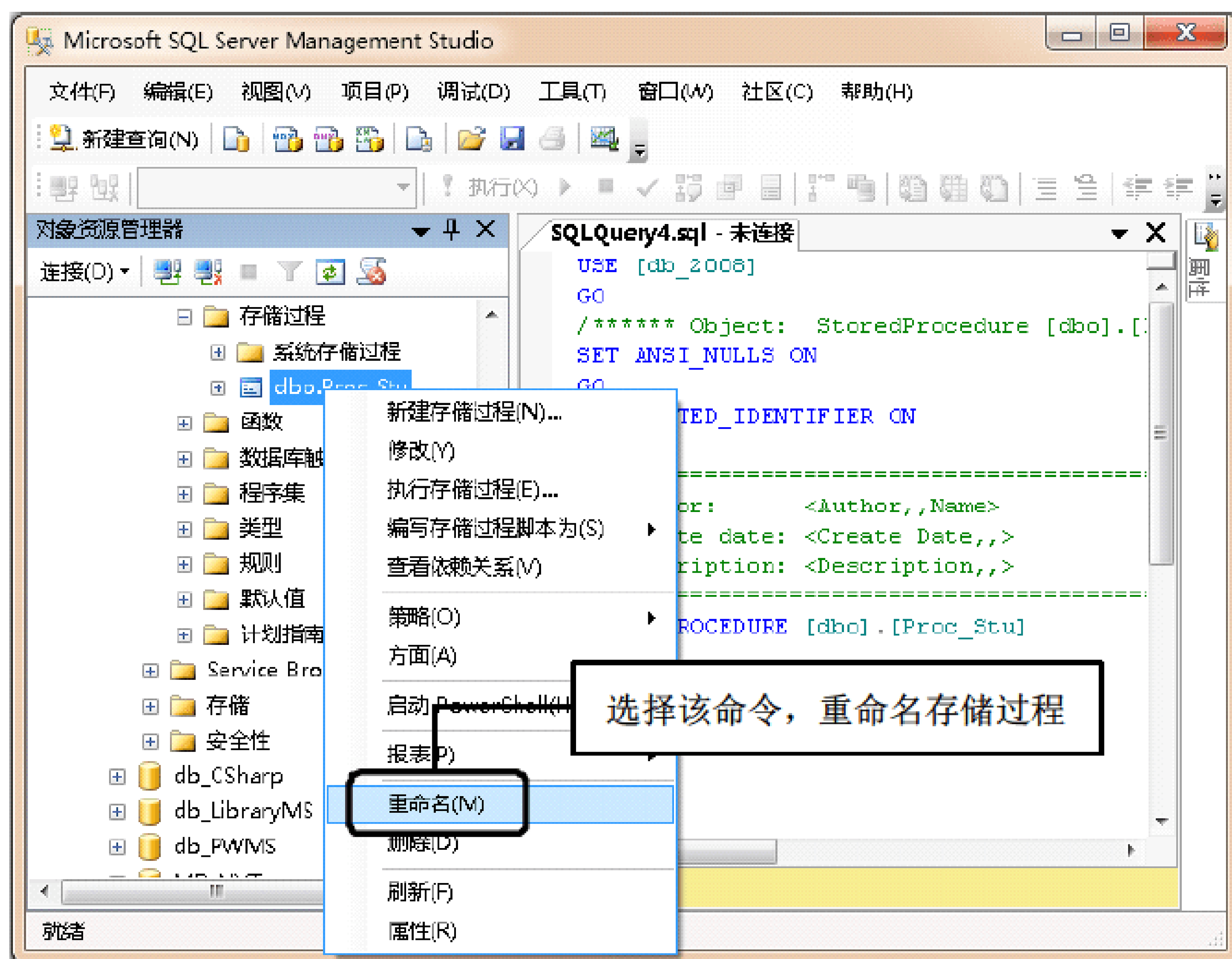


图 13.11　重命名存储过程

（3）此时，在存储过程名称的文本框中输入要修改的名称，即可重命名存储过程。

2. 执行 sp_rename 系统存储过程重新命名存储过程

sp_rename 系统存储过程可以在当前数据库中更改用户创建对象的名称。此对象可以是表、索引、列、别名数据类型或 Microsoft .NET Framework 公共语言运行时（CLR）用户定义类型。语法格式如下：

```
sp_rename [@objname =] 'object_name' , [@newname =] 'new_name'
    [, [@objtype =] 'object_type']
```

参数说明如下。

☑ [@objname =] 'object_name'：用户对象或数据类型的当前限定或非限定名称。如果要重命名的对象是表中的列，则 object_name 的格式必须是 table.column。如果要重命名的对象是索引，则 object_name 的格式必须是 table.index。

☑ [@newname =] 'new_name'：指定对象的新名称。new_name 必须是名称的一部分，并且必须遵循标识符的规则。newname 的数据类型为 sysname，无默认值。

☑ [@objtype =] 'object_type'：要重命名的对象的类型。

使用 sp_rename 系统存储过程重新命名存储过程的步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）选择需要重新命名的存储过程所在的数据库，单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，输入执行 sp_rename 系统存储过程重新命名的 SQL 语句。

【例 13.07】　将 Proc_Stu 存储过程重新命名为 Proc_StuInfo。（**实例位置：资源包\源码\13\13.07**）

SQL 语句如下：

```
sp_rename 'Proc_Stu','Proc_StuInfo'
```

（3）单击 ! 执行(X) 按钮，就可以执行上述 SQL 语句代码。结果如图 13.12 所示。

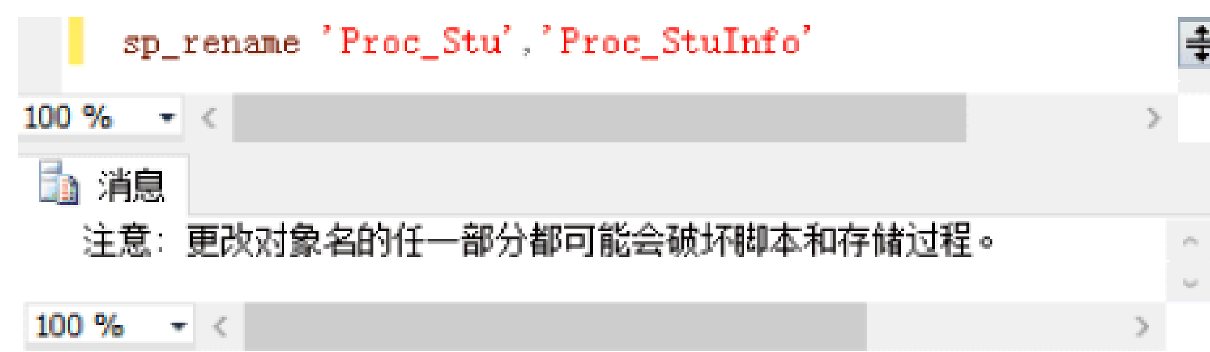


图 13.12 重新命名的存储过程


注意

更改对象名的任一部分都可能破坏脚本和存储过程。建议读者不要使用此语句来重命名存储过程、触发器、用户定义函数或视图；而是删除该对象，然后使用新名称重新创建该对象。

13.3.5 删除存储过程

数据库中某些不再应用的存储过程可以将其删除，这样节约该存储过程所占的数据库空间。删除存储过程可以通过手动删除或执行 DROP PROCEDURE 语句实现。

1. 手动删除存储过程

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）展开“对象资源管理器”中“数据库”→“数据库名称”→“可编程性”→“存储过程”节点，鼠标右键单击要删除的存储过程，在弹出的快捷菜单中选择“删除”命令。

（3）在弹出的“删除对象”窗口中确认所删除的存储过程，单击“确定”按钮即可将该存储过程删除。

例如，删除 Proc_StuInfo 存储过程，如图 13.13 所示。

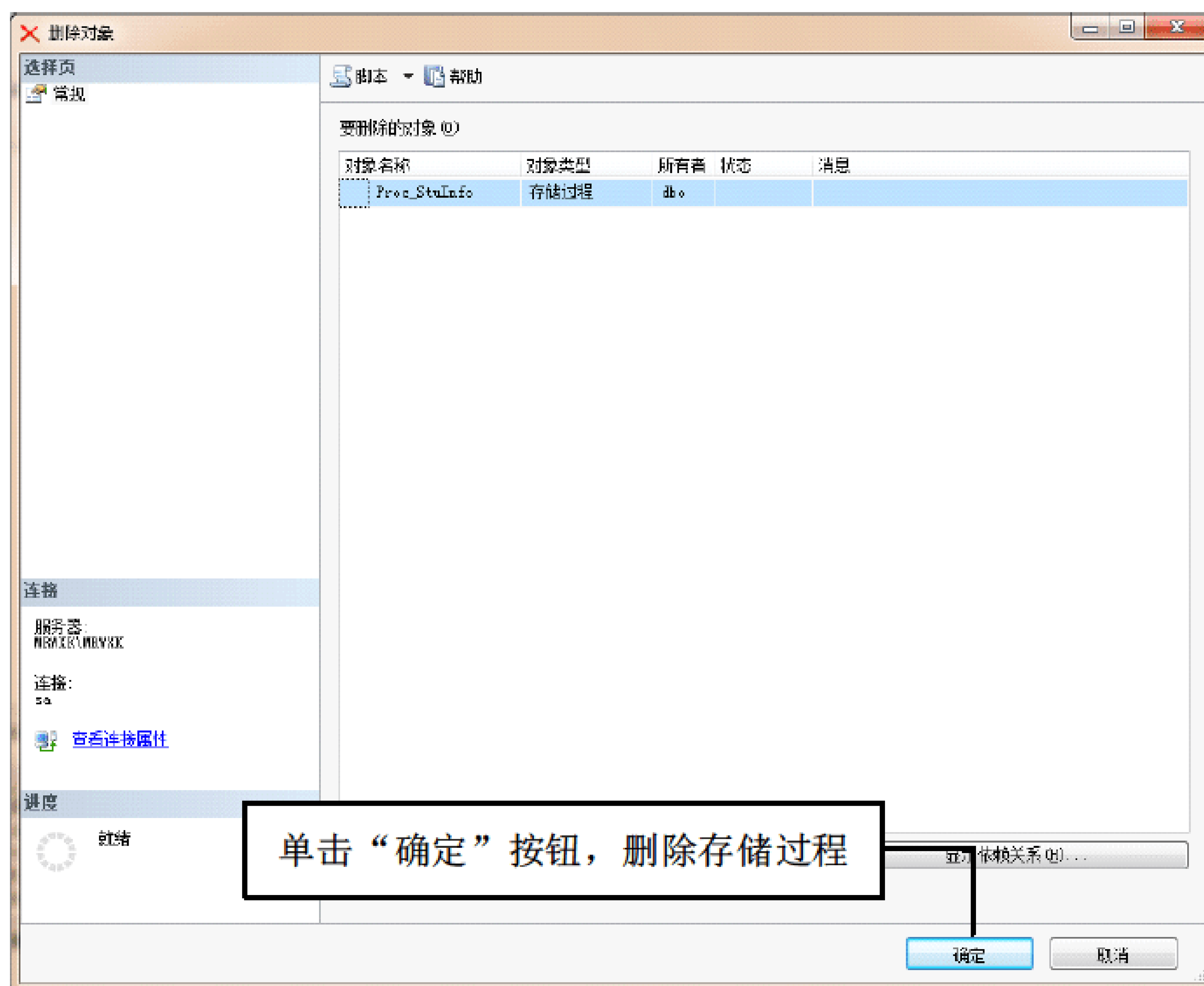


图 13.13 删除存储过程

2. 执行 DROP PROCEDURE 语句删除存储过程

DROP PROCEDURE 语句用来从当前数据库中删除一个或多个存储过程。语法格式如下：

```
DROP {PROC | PROCEDURE} {[schema_name.] procedure} [,...n]
```

参数说明如下。

☑ schema_name：过程所属架构的名称。不能指定服务器名称或数据库名称。

☑ procedure：要删除的存储过程或存储过程组的名称。

执行 DROP PROCEDURE 语句删除存储过程的步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

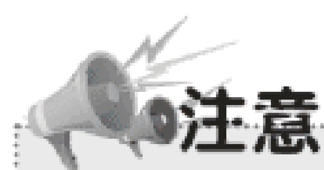
（2）选择需要删除的存储过程所在的数据库，单击工具栏中的新建查询(N)按钮，在新建查询编辑器中输入执行 DROP PROCEDURE 语句删除存储过程的 SQL 语句。

【例 13.08】 删除名为 Proc_Student 的存储过程。（**实例位置：资源包\源码\13\13.08**）

SQL 语句如下：

```
DROP PROCEDURE Proc_Student
```

（3）单击执行(X)按钮，就可以执行上述 SQL 语句代码，将 Proc_Student 存储过程删除。



注意

不可以删除正在使用的存储过程，否则 Microsoft SQL Server 2014 将在执行调用进程时显示一条错误消息。

13.4 小　　结

本章介绍了存储过程的概念，以及创建和管理存储过程的方法。读者使用存储过程可以增强代码的重用性，创建存储过程后可以调用 EXECUTE 语句执行存储过程或者设置其自动执行，另外，还可以查看、修改或者删除存储过程，使读者能够更容易理解存储过程。

第14章

触发器

（视频讲解：11分钟）

本章主要介绍如何使用触发器，包括触发器概述、创建触发器、修改触发器和删除触发器等内容。通过本章的学习，读者可以掌握使用Transact-SQL创建触发器，并应用触发器编写SQL语句，从而优化查询和提高数据访问速度。

学习摘要：

- 触发器的基本概念
- 触发器的分类及创建
- 使用sp_helptext存储过程查看触发器
- 创建DML、DDL触发器
- 使用sp_rename重命名触发器
- 禁用和启用触发器
- 删除触发器


视频讲解

14.1 触发器概述

14.1.1 触发器的概念

Microsoft SQL Server 提供两种主要机制来强制使用业务规则和数据完整性：约束和触发器。

触发器是一种特殊类型的存储过程，当指定表中的数据发生变化时触发器自动生效。它与表紧密相连，可以看作是表定义的一部分。触发器不能通过名称被直接调用，更不允许设置参数。

在 SQL Server 中一张表可以有多个触发器。用户可以使用 INSERT、UPDATE 或 DELETE 语句对触发器进行设置，也可以对一张表上的特定操作设置多个触发器。触发器可以包含复杂的 Transact-SQL 语句。不论触发器所进行的操作有多复杂，触发器都只作为一个独立的单元被执行，被看作是一个事务。如果在执行触发器的过程中发生了错误，则整个事务将会自动回滚。

14.1.2 触发器的优点

触发器的优点表现在以下几个方面。

（1）触发器自动执行，对表中的数据进行修改后，触发器立即被激活。

（2）为了实现复杂的数据库更新操作，触发器可以调用一个或多个存储过程，甚至可以通过调用外部过程（不是数据库管理系统本身）完成相应的操作。

（3）触发器能够实现比 CHECK 约束更为复杂的数据完整性约束。在数据库中，为了实现数据完整性约束，可以使用 CHECK 约束或触发器。CHECK 约束不允许引用其他表中的列来完成检查工作，而触发器可以引用其他表中的列。它更适合在大型数据库管理系统中用来约束数据的完整性。

（4）触发器可以检测数据库内的操作，从而取消了数据库未经许可的更新操作，使数据库修改、更新操作更安全，数据库的运行也更稳定。

（5）触发器能够对数据库中的相关表实现级联更改。触发器是基于一个表创建的，但是可以针对多个表进行操作，实现数据库中相关表的级联更改。

（6）一个表中可以同时存在 3 个不同操作的触发器（INSERT、UPDATE 和 DELETE）。

14.1.3 触发器的种类

SQL Server 包括 3 种常规类型的触发器：DML 触发器、DDL 触发器和登录触发器。

当数据库中发生数据操作语言（DML）事件时将调用 DML 触发器。DML 事件包括在指定表或视图中修改数据的 INSERT 语句、UPDATE 语句或 DELETE 语句。DML 触发器可以查询其他表，还可以包含复杂的 Transact-SQL 语句。

读者可以设计以下类型的 DML 触发器。

☑ AFTER 触发器：在执行了 INSERT、UPDATE 或 DELETE 语句操作之后执行 AFTER 触发器。

☑ INSTEAD OF 触发器：执行 INSTEAD OF 触发器代替通常的触发动作。还可为带有一个或多个基表的视图定义 INSTEAD OF 触发器，而这些触发器能够扩展视图可支持的更新类型。

☑ CLR 触发器：可以是 AFTER 触发器或 INSTEAD OF 触发器。CLR 触发器还可以是 DDL 触发器。CLR 触发器将执行在托管代码（在.NET Framework 中创建并在 SQL Server 中上载的程序集的成员）中编写的方法，而不用执行 Transact-SQL 存储过程。

DDL 触发器是一种特殊的触发器，它在响应数据定义语言（DDL）语句时触发，可以用于在数据库中执行管理任务，如审核以及规范数据库操作。

登录触发器将为响应 LOGON 事件而激发存储过程。与 SQL Server 实例建立用户会话时将引发此事件。登录触发器将在登录的身份验证阶段完成之后且用户会话实际建立之前激发。可以使用登录触发器来审核和控制服务器会话，例如，通过跟踪登录活动，限制 SQL Server 的登录名或限制特定登录名的会话数。

14.2　创建触发器


视频讲解

创建 DML 触发器、DDL 触发器和登录触发器可以通过执行 CREATE TRIGGER 语句实现。但在使用该语句创建 DML 触发器、DDL 触发器和登录触发器时，其语法存在差异。本节讲解 CREATE TRIGGER 语句与使用该语句创建 DML 触发器、DDL 触发器和登录触发器。

14.2.1　创建 DML 触发器

如果用户要通过数据操作语言（DML）事件编辑数据，则执行 DML 触发器。DML 事件是针对表或视图的 INSERT、UPDATE 或 DELETE 语句。

创建 DML 触发器的语法格式如下：

```
CREATE TRIGGER [schema_name .]trigger_name
ON {table | view}
[WITH <dml_trigger_option> [,...n]]
{FOR | AFTER | INSTEAD OF}
{[INSERT] [,] [UPDATE] [,] [DELETE]}
[WITH APPEND]
[NOT FOR REPLICATION]
AS {sql_statement   [;] [,...n] | EXTERNAL NAME <method specifier [;] >}
<dml_trigger_option> ::=
    [ENCRYPTION]
    [EXECUTE AS Clause]
<method_specifier> ::=
    assembly_name.class_name.method_name
```

创建 DML 触发器的参数及说明如表 14.1 所示。

表 14.1　创建 DML 触发器的参数及说明

参　　数	描　　述
schema_name	DML 触发器所属架构的名称。DML 触发器的作用域是为其创建该触发器的表或视图的架构
trigger_name	触发器的名称。trigger_name 必须遵循标识符规则，但 trigger_name 不能以#或##开头
table | view	对其执行 DML 触发器的表或视图，有时称为触发器表或触发器视图。可以根据需要指定表或视图的完全限定名称。视图只能被 INSTEAD OF 触发器引用。不能对局部或全局临时表定义 DML 触发器
FOR | AFTER	AFTER 指定 DML 触发器仅在触发 SQL 语句中指定的所有操作都已成功执行时才被触发
INSTEAD OF	指定执行 DML 触发器而不是触发 SQL 语句，因此，其优先级高于触发语句的操作
{[INSERT] [,] [UPDATE] [,] [DELETE]}	指定数据修改语句，这些语句可在 DML 触发器对此表或视图进行尝试时激活该触发器。必须至少指定一个选项
WITH APPEND	指定应该再添加一个现有类型的触发器
NOT FOR REPLICATION	指示当复制代理修改涉及触发器的表时，不应执行触发器
sql_statement	触发条件和操作。触发器条件指定其他标准，用于确定尝试的 DML、DDL 或 LOGON 事件是否导致执行触发器操作
EXECUTE AS	指定用于执行该触发器的安全上下文
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法。该方法不能带有任何参数，并且必须返回空值

【例 14.01】 为员工表 employee3 创建 DML 触发器，当向该表中插入数据时给出提示信息。（**实例位置：资源包\源码\14\14.01**）

设计步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，输入如下 SQL 语句代码。

```
CREATE TRIGGER T_DML_Emp3                              --创建触发器 T_DML_Emp3
ON employee3                                           --依赖于表 employee3
AFTER INSERT                                           --执行插入语句之后
AS
RAISERROR ('正在向表中插入数据', 16, 10);              --提示信息
```

（3）单击 执行(X) 按钮，执行上述 SQL 语句代码，创建名称为 T_DML_Emp3 的 DML 触发器。

每次对 employee3 表的数据进行添加时，都会显示如图 14.1 所示的消息内容。



图 14.1　向表中插入数据时给出的信息

14.2.2　创建 DDL 触发器

DDL 触发器用于响应各种数据定义语言（DDL）事件。这些事件主要对应于 Transact-SQL 的 CREATE、

ALTER 和 DROP 语句，以及执行类似 DDL 操作的某些系统存储过程。

创建 DDL 触发器的语法格式如下：

```
CREATE TRIGGER trigger_name
ON {ALL SERVER | DATABASE}
[WITH <ddl_trigger_option> [,...n]]
{FOR | AFTER} {event_type | event_group} [,...n]
AS {sql_statement   [;] [,...n] | EXTERNAL NAME <method specifier>   [;]}
<ddl_trigger_option> ::=
    [ENCRYPTION]
    [EXECUTE AS Clause]
<method_specifier> ::=
    assembly_name.class_name.method_name
```

创建 DDL 触发器的参数及说明如表 14.2 所示。

表 14.2　创建 DDL 触发器的参数及说明

参　数	描　述
trigger_name	触发器的名称。trigger_name 必须遵循标识符规则，但 trigger_name 不能以#或##开头
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
DATABASE	将 DDL 触发器的作用域应用于当前数据库
FOR | AFTER	AFTER 指定 DML 触发器仅在触发 SQL 语句中指定的所有操作都已成功执行时才被触发
event_type	执行之后将导致激发 DDL 触发器的 Transact-SQL 语言事件的名称。DDL 事件中列出了 DDL 触发器的有效事件
event_group	预定义的 Transact-SQL 语言事件分组的名称
sql_statement	触发条件和操作。触发器条件指定其他标准，用于确定尝试的 DML、DDL 或 LOGON 事件是否导致执行触发器操作
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法

【例 14.02】　为数据库 db_2014 创建 DDL 触发器，防止用户对表进行删除或修改等操作。（**实例位置：资源包\源码\14\14.02**）

设计步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，输入如下 SQL 语句代码。

```
CREATE TRIGGER T_DDL_DATABASE                --创建 DDL 触发器
ON DATABASE                                  --将该触发器应用于当前数据库
FOR DROP_TABLE, ALTER_TABLE                  --对表修改时，提示信息
AS
PRINT '只有“T_DDL_DATABASE”触发器无效时，才可以删除或修改表。'
ROLLBACK                                     --回滚操作
```

（3）单击 ! 执行(X) 按钮，执行上述 SQL 语句代码。创建名称为 T_DDL_DATABASE 的 DDL 触发器。

创建完该触发器后，当对数据库中的表进行修改与删除等操作时，都会提示：只有“T_DDL_DATABASE”触发器无效时，才可以删除或修改表。并将删除后修改操作进行回滚。显示信息如图 14.2

所示。

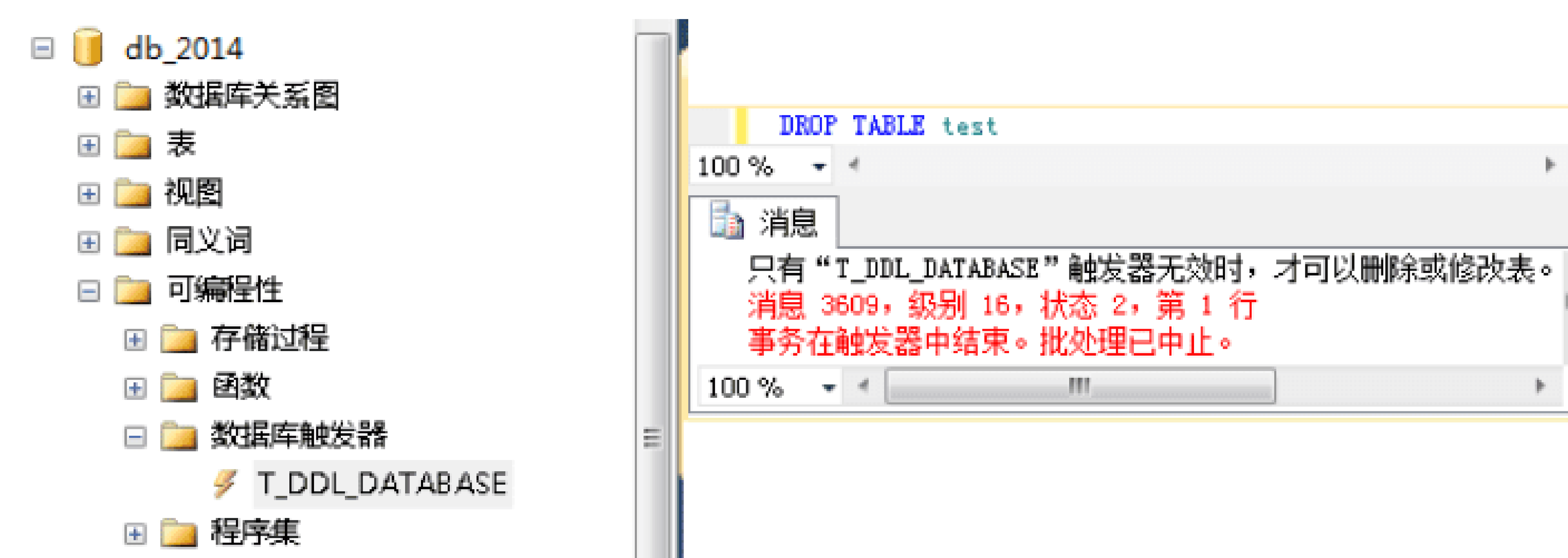


图 14.2　对数据库中表进行修改与删除等操作时显示的消息

14.2.3　创建登录触发器

登录触发器在遇到 LOGON 事件时触发。LOGON 事件是在建立用户会话时引发的。触发器可以由 Transact-SQL 语句直接创建，也可以由程序集方法创建，这些方法是在 Microsoft .NET Framework 公共语言运行时（CLR）中创建并上载到 SQL Server 实例的。SQL Server 允许为任何特定语句创建多个触发器。

创建登录触发器的语法格式如下：

```
CREATE TRIGGER trigger_name
ON ALL SERVER
[WITH <logon_trigger_option> [,...n]]
{FOR | AFTER} LOGON
AS {sql_statement   [;] [,...n] | EXTERNAL NAME <method specifier>   [;]}
<logon_trigger_option> ::=
    [ENCRYPTION]
    [EXECUTE AS Clause]
<method_specifier> ::=
    assembly_name.class_name.method_name
```

创建登录触发器的参数及说明如表 14.3 所示。

表 14.3　创建登录触发器的参数及说明

参　　数	描　　述
trigger_name	触发器的名称。trigger_name 必须遵循标识符规则，但 trigger_name 不能以#或##开头
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
FOR | AFTER	AFTER 指定 DML 触发器仅在触发 SQL 语句中指定的所有操作都已成功执行时才被触发
sql_statement	触发条件和操作。触发器条件指定其他标准，用于确定尝试的 DML、DDL 或 LOGON 事件是否导致执行触发器操作
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法

【例 14.03】　创建一个登录触发器，该触发器拒绝登录名为 mr 的成员登录 SQL Server。（**实例位置：资源包\源码\14\14.03**）

SQL 语句如下：

```
USE master;
GO
CREATE LOGIN TM WITH PASSWORD = 'TMsoft' MUST_CHANGE,
    CHECK_EXPIRATION = ON;
GO
GRANT VIEW SERVER STATE TO TM;
GO
CREATE TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS 'mr'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= 'mr' AND
    (SELECT COUNT(*) FROM sys.dm_exec_sessions
            WHERE is_user_process = 1 AND
                original_login_name = 'mr') > 1
    ROLLBACK;
END;
```

设计步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）单击工具栏中的新建查询(N)按钮，新建查询编辑器，输入例 14.03 中的 SQL 语句。

（3）单击执行(X)按钮，执行上述 SQL 语句代码。创建名称为 connection_limit_trigger 的登录触发器。

登录触发器与 DML 触发器、DDL 触发器所存储的位置不同，其存储位置为“对象资源管理器”中的“服务器对象”→“触发器”。登录触发器 connection_limit_trigger 中的 mr 为登录到 SQL Server 中的登录名。触发器及 mr 所在的位置如图 14.3 所示。

创建完该触发器后，当以登录名 mr 登录 SQL Server 时，就会显示如图 14.4 所示的提示信息。



图 14.3　触发器及 mr 所在的位置



图 14.4　登录名 mr 登录 SQL Server 时提示的信息



14.3　管理触发器

触发器的查看、修改、重命名、禁用和启用，以及删除等操作都可以使用 SQL Server Management Studio 管理工具实现。本节讲解通过 SQL 命令管理工具查看、修改、重命名、禁用和启用，以及删除触发器。

14.3.1　查看触发器

查看触发器与查看存储过程相同。同样可以使用 sp_helptext 存储过程与 sys.sql_modules 视图查看触发器。

1．使用 sp_helptext 存储过程查看触发器

sp_helptext 存储过程可以查看架构范围内的触发器，非架构范围内的触发器是不能用此存储过程查看的，如 DDL 触发器、登录触发器。

【例 14.04】　sp_helptext 存储过程查看 DML 触发器，SQL 语句及运行结果如图 14.5 所示。（**实例位置：资源包\源码\14\14.04**）

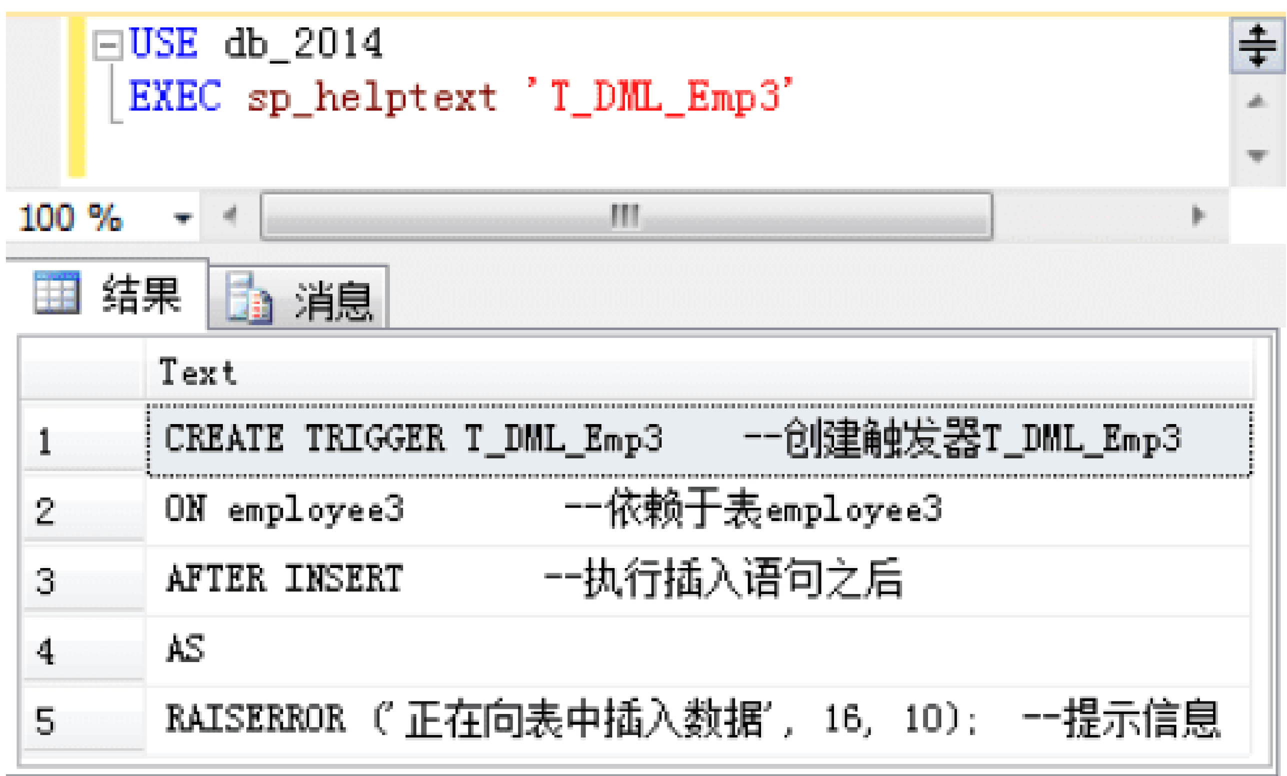


图 14.5　使用 sp_helptext 存储过程查看 DML 触发器

SQL 语句如下：

```
USE db_2014
EXEC sp_helptext 'T_DML_Emp3'
```

2．获取数据库中触发器的信息

每个类型为 TR 或 TA 的触发器对象对应一行，TA 代表程序集（CLR）触发器，TR 代表 SQL 触发器。DML 触发器名称在架构范围内，因此，可在 sys.objects 中显示。DDL 触发器名称的作用域取决于父实体，只能在对象目录视图中显示。

【例 14.05】　在 db_2014 数据库中，查找类型为 TR 的触发器，即 DDL 触发器，SQL 语句及运行结果如图 14.6 所示。（**实例位置：资源包\源码\14\14.05**）

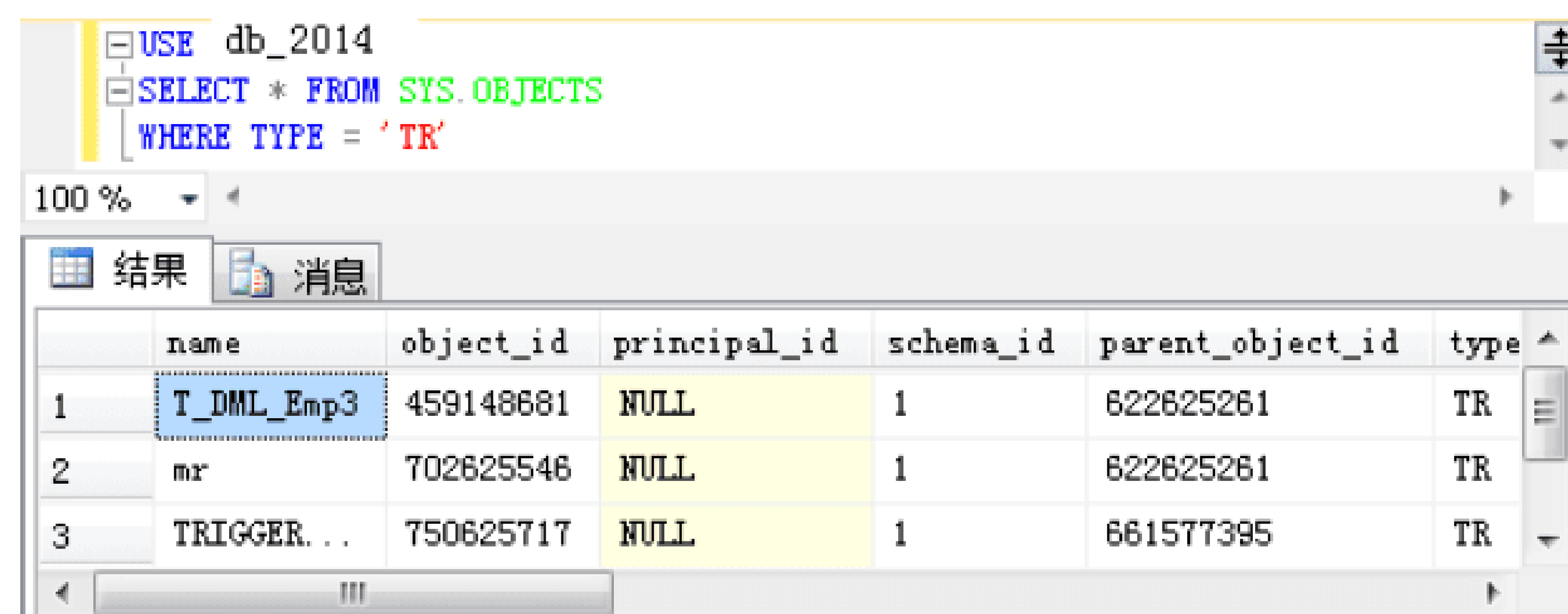


图 14.6 查找 DDL 触发器

SQL 语句如下：

```
USE db_2014
SELECT * FROM sys.objects
WHERE TYPE='TR'
```

14.3.2 修改触发器

修改触发器可以通过 ALTER TRIGGER 语句实现，下面分别对修改 DML 触发器、修改 DDL 触发器、修改登录触发器进行介绍。

1．修改 DML 触发器

修改 DML 触发器的语法格式如下：

```
ALTER TRIGGER schema_name.trigger_name
ON (table | view)
[WITH <dml_trigger_option> [,...n]]
(FOR | AFTER | INSTEAD OF)
{[DELETE] [,] [INSERT] [,] [UPDATE]}
[NOT FOR REPLICATION]
AS {sql_statement [;] [...n] | EXTERNAL NAME <method specifier> [;]}
<dml_trigger_option> ::=
    [ENCRYPTION]
    [<EXECUTE AS Clause>]
<method_specifier> ::=
        assembly_name.class_name.method_name
```

修改 DML 触发器的参数及说明如表 14.4 所示。

表 14.4 修改 DML 触发器的参数及说明

参　　数	描　　述
schema_name	DML 触发器所属架构的名称。DML 触发器的作用域是为其创建该触发器的表或视图的架构
trigger_name	要修改的现有触发器
table | view	对其执行 DML 触发器的表或视图，有时称为触发器表或触发器视图。可以根据需要指定表或视图的完全限定名称

续表

参　　数	描　　述
AFTER	指定只有在触发 SQL 语句成功执行后，才会激发触发器
INSTEAD OF	指定执行 DML 触发器而不是触发 SQL 语句，因此，其优先级高于触发语句的操作
{[DELETE] [,] [INSERT] [,] [UPDATE]}	指定数据修改语句在试图修改表或视图时，激活 DML 触发器。必须至少指定一个选项
NOT FOR REPLICATION	指示当复制代理修改涉及触发器的表时，不应执行触发器
sql_statement	触发条件和操作
EXECUTE AS	指定用于执行该触发器的安全上下文
< method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法。该方法不能带有任何参数，并且必须返回空值

【例 14.06】　使用 ALTER TRIGGER 语句修改 DML 触发器 T_DML_Emp3，当向该表中插入、修改或删除数据时给出提示信息。（**实例位置：资源包\源码\14\14.06**）

SQL 语句如下：

```
ALTER TRIGGER T_DML_Emp3
ON employee3
AFTER INSERT,UPDATE,DELETE
AS
RAISERROR ('正在向表中插入、修改或删除数据', 16, 10);
```

运行结果如图 14.7 所示。

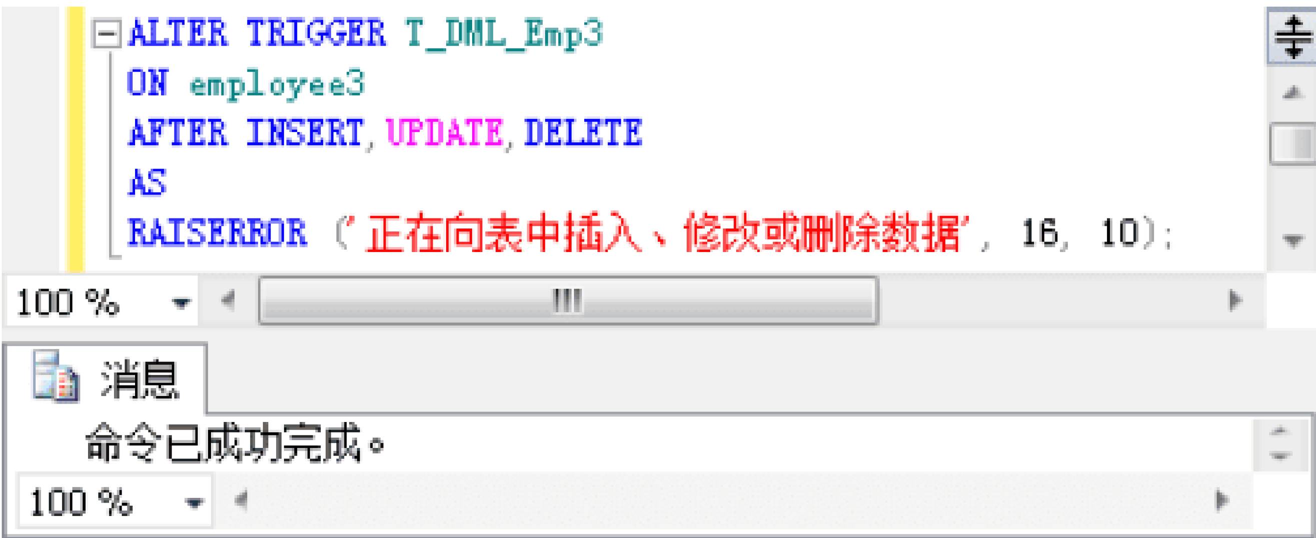


图 14.7　使用 ALTER TRIGGER 修改 DML 触发器

2．修改 DDL 触发器

修改 DDL 触发器的语法格式如下：

```
ALTER TRIGGER trigger_name
ON {DATABASE | ALL SERVER}
[WITH <ddl_trigger_option> [,...n]]
{FOR | AFTER} {event_type [,...n] | event_group}
AS {sql_statement [;] | EXTERNAL NAME <method specifier>
[;]}
}
<ddl_trigger_option> ::=
    [ENCRYPTION]
    [<EXECUTE AS Clause>]
```

```
<method_specifier> ::=
        assembly_name.class_name.method_name
```

修改 DDL 触发器的参数及说明如表 14.5 所示。

表 14.5　修改 DDL 触发器的参数及说明

参　数	描　述
trigger_name	要修改的现有触发器
DATABASE	将 DDL 触发器的作用域应用于当前数据库
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
AFTER	指定只有在触发 SQL 语句成功执行后，才会激发触发器
event_type	执行之后将导致激发 DDL 触发器的 Transact-SQL 语言事件的名称
event_group	预定义的 Transact-SQL 语言事件分组的名称
sql_statement	触发条件和操作
EXECUTE AS	指定用于执行该触发器的安全上下文
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法。该方法不能带有任何参数，并且必须返回空值

【例 14.07】 使用 ALTER TRIGGER 语句修改 DDL 触发器 T_DDL_DATABASE，防止用户修改数据。（**实例位置：资源包\源码\14\14.07**）

SQL 语句如下：

```
ALTER TRIGGER T_DDL_DATABASE                    --修改触发器
ON DATABASE                                     --应用于当前数据库
FOR ALTER_TABLE
AS
RAISERROR ('只有“T_DDL_DATABASE”触发器无效时，才可以修改表。', 16, 10)
ROLLBACK                                        --回滚事务
```

3．修改登录触发器

修改登录触发器的语法格式如下：

```
ALTER TRIGGER trigger_name
ON ALL SERVER
[WITH <logon_trigger_option> [,...n]]
{FOR | AFTER} LOGON
AS {sql_statement   [;] [,...n] | EXTERNAL NAME < method specifier >   [;]}
<logon_trigger_option> ::=
    [ENCRYPTION]
    [EXECUTE AS Clause]
<method_specifier> ::=
    assembly_name.class_name.method_name
```

修改登录触发器的参数及说明如表 14.6 所示。

表 14.6　修改登录触发器的参数及说明

参　　数	描　　述
trigger_name	要修改的现有触发器
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
AFTER	指定只有在触发 SQL 语句成功执行后，才会激发触发器
sql_statement	触发条件和操作
EXECUTE AS	指定用于执行该触发器的安全上下文
<method_specifier>	指定要与触发器绑定的程序集的方法

【例 14.08】　使用 ALTER TRIGGER 语句修改登录触发器 connection_limit_trigger，将用户名修改为 nxt，如果在此登录名下已运行 3 个用户会话，拒绝 nxt 登录到 SQL Server。（**实例位置：资源包\源码\14\14.08**）

SQL 语句如下：

```
ALTER TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS 'nxt'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= 'nxt' AND
    (SELECT COUNT(*) FROM sys.dm_exec_sessions
            WHERE is_user_process = 1 AND
                original_login_name = 'nxt') > 3
    ROLLBACK;
END;
```

14.3.3　重命名触发器

重命名触发器可以使用 sp_rename 系统存储过程实现。使用 sp_rename 系统存储过程重命名触发器与重命名存储过程相同。但是使用该系统存储过程重命名触发器，不会更改 sys.sql_modules 类别视图的 definition（用于定义此模块的 SQL 文本）列中相应对象名的名称，所以建议用户不要使用该系统存储过程重命名触发器，而是删除该触发器，然后使用新名称重新创建该触发器。

【例 14.09】　使用 sp_rename 将触发器 T_DML_Emp3 重命名为 T_DML_3。（**实例位置：资源包\源码\14\14.09**）

SQL 语句如下：

```
sp_rename 'T_DML_Emp3','T_DML_3'
```

14.3.4　禁用和启用触发器

当不再需要某个触发器时，可将其禁用或删除。禁用触发器不会删除该触发器，该触发器仍然作

为对象存在于当前数据库中。但是，当执行任意 INSERT、UPDATE 或 DELETE 语句（在其上对触发器进行了编程）时，触发器将不会激发。已禁用的触发器可以被重新启用。启用触发器会以最初创建它时的方式将其激发。默认情况下，创建触发器后会启用触发器。

1．禁用触发器

使用 DISABLE TRIGGER 语句禁用触发器，其语法格式如下：

```
DISABLE TRIGGER {[schema_name .] trigger_name [,...n] | ALL}
ON {object_name | DATABASE | ALL SERVER} [;]
```

参数说明如下。

☑ schema_name：触发器所属架构的名称。

☑ trigger_name：要禁用的触发器的名称。

☑ ALL：指示禁用在 ON 子句作用域中定义的所有触发器。



注意

SQL Server 在为合并复制发布的数据库中创建触发器。在已发布数据库中指定 ALL 可禁用这些触发器，这样会中断复制。在指定 ALL 之前，请验证没有为合并复制发布当前数据库。

☑ object_name：要对其创建要执行的 DML 触发器 trigger_name 的表或视图的名称。

☑ DATABASE：对于 DDL 触发器，指示所创建或修改的 trigger_name 将在数据库范围内执行。

☑ ALL SERVER：对于 DDL 触发器，指示所创建或修改的 trigger_name 将在服务器范围内执行。ALL SERVER 也适用于登录触发器。

【例 14.10】 使用 DISABLE TRIGGER 语句禁用 DML 触发器 T_DML_3。（**实例位置：资源包\源码\14\14.10**）

SQL 语句如下：

```
DISABLE TRIGGER T_DML_3 ON employee3
```

禁用后触发器的状态如图 14.8 所示。



图 14.8　禁用触发器的状态

【例 14.11】 使用 DISABLE TRIGGER 语句禁用 DDL 触发器 T_DDL_DATABASE。（**实例位置：资源包\源码\14\14.11**）

SQL 语句如下：

```
DISABLE TRIGGER T_DDL_DATABASE ON DATABASE
```

【例 14.12】 使用 DISABLE TRIGGER 语句禁用登录触发器 connection_limit_trigger。（**实例位置：资源包\源码\14\14.12**）

SQL 语句如下：

```
DISABLE TRIGGER connection_limit_trigger ON ALL SERVER
```

2. 启用触发器

启用触发器并不是重新创建它。已禁用的 DDL 触发器、DML 触发器或登录触发器可以通过执行 ENABLE TRIGGER 语句重新起用。语法格式如下：

```
ENABLE TRIGGER {[schema_name .] trigger_name [,...n] | ALL}
ON {object_name | DATABASE | ALL SERVER} [;]
```

启用触发器的参数及说明如表 14.7 所示。

表 14.7　启用触发器的参数及说明

参　　数	描　　述
schema_name	触发器所属架构的名称。不能为 DDL 或登录触发器指定 schema_name
trigger_name	要启用的触发器的名称
ALL	指示启用在 ON 子句作用域中定义的所有触发器
object_name	要对其创建要执行的 DML 触发器 trigger_name 的表或视图的名称
DATABASE	对于 DDL 触发器，指示所创建或修改的 trigger_name 将在数据库范围内执行
ALL SERVER	对于 DDL 触发器，指示所创建或修改的 trigger_name 将在服务器范围内执行。ALL SERVER 也适用于登录触发器

【例 14.13】 使用 ENABLE TRIGGER 语句启用 DML 触发器 T_DML_3。（**实例位置：资源包\源码\14\14.13**）

SQL 语句如下：

```
ENABLE TRIGGER T_DML_3 on employee3
```

【例 14.14】 使用 ENABLE TRIGGER 语句启用 DDL 触发器 T_DDL_DATABASE。（**实例位置：资源包\源码\14\14.14**）

SQL 语句如下：

```
ENABLE TRIGGER T_DDL_DATABASE ON DATABASE
```

【例 14.15】 使用 ENABLE TRIGGER 语句启用登录触发器 connection_limit_trigger。（**实例位置：资源包\源码\14\14.15**）

SQL 语句如下：

```
ENABLE TRIGGER connection_limit_trigger ON ALL SERVER
```

启用后触发器的状态如图 14.9 所示。

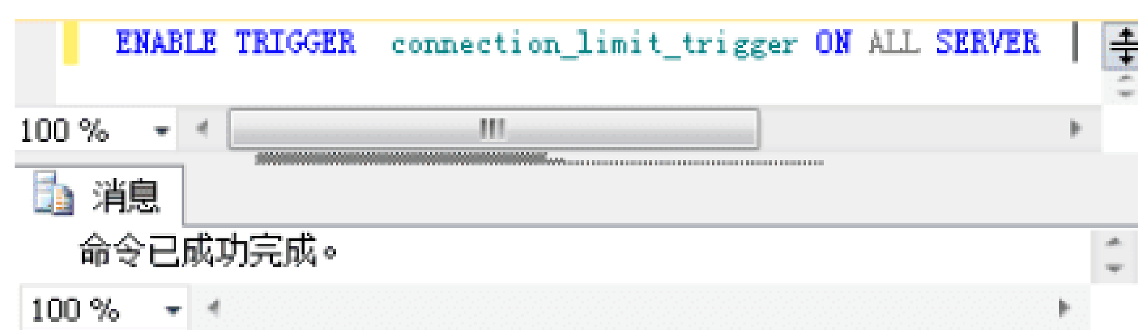


图 14.9　启用后触发器的状态

14.3.5　删除触发器

删除触发器是将触发器对象从当前数据库中永久地删除。通过执行 DROP TRIGGER 语句可以将 DML 触发器、DDL 触发器或登录触发器删除。也可以通过操作 SQL Server Management Studio 手动删除 DML 触发器、DDL 触发器或登录触发器。

1．DROP TRIGGER 语句删除触发器

DROP TRIGGER 语句可以从当前数据库中删除一个或多个 DML 触发器、DDL 触发器或登录触发器。

（1）删除 DML 触发器

删除 DML 触发器的语法格式如下：

```
DROP TRIGGER schema_name.trigger_name [,...n] [;]
```

参数说明如下。

☑ schema_name：DML 触发器所属架构的名称。

☑ trigger_name：要删除的触发器的名称。

【例 14.16】　使用 DROP TRIGGER 语句删除 DML 触发器 T_DML_3。（**实例位置：资源包\源码\14\14.16**）

SQL 语句如下：

```
DROP TRIGGER T_DML_3
```

（2）删除 DDL 触发器

删除 DDL 触发器的语法格式如下：

```
DROP TRIGGER trigger_name [,...n]
ON {DATABASE | ALL SERVER}
[;]
```

参数说明如下。

☑ trigger_name：要删除的触发器的名称。

☑ DATABASE：指示 DDL 触发器的作用域应用于当前数据库。如果在创建或修改触发器时也指定了 DATABASE，则必须指定 DATABASE。

☑ ALL SERVER：指示 DDL 触发器的作用域应用于当前服务器。如果在创建或修改触发器时也指定了 ALL SERVER，则必须指定 ALL SERVER。ALL SERVER 也适用于登录触发器。

【例 14.17】　使用 DROP TRIGGER 语句删除 DDL 触发器 T_DDL_DATABASE。（**实例位置：资源包\源码\14\14.17**）

SQL 语句如下：

```
DROP TRIGGER T_DDL_DATABASE   ON DATABASE
```

（3）删除登录触发器

删除登录触发器的语法格式如下：

```
DROP TRIGGER trigger_name [,...n]
ON ALL SERVER
```

参数说明如下。

☑ trigger_name：要删除的触发器的名称。

☑ ALL SERVER：将 DDL 触发器或登录触发器的作用域应用于当前服务器。如果指定了此参数，则只要当前服务器中的任何位置上出现 event_type 或 event_group，就会激发该触发器。

【例 14.18】 使用 DROP TRIGGER 语句删除登录触发器 connection_limit_trigger。（**实例位置：资源包\源码\14\14.18**）

SQL 语句如下：

```
DROP TRIGGER connection_limit_trigger ON ALL SERVER
```

2．SQL Server Management Studio 手动删除触发器

手动删除触发器步骤如下。

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。

（2）展开“对象资源管理器”中触发器所在位置。例如，要删除创建在 db_2014 数据库的 T_DDL_DATABASE 触发器，则展开如图 14.10 所示的树型结构。

（3）鼠标右键单击要删除的触发器，在弹出的快捷菜单中选择“删除”命令，打开“删除对象”窗口，如图 14.11 所示。

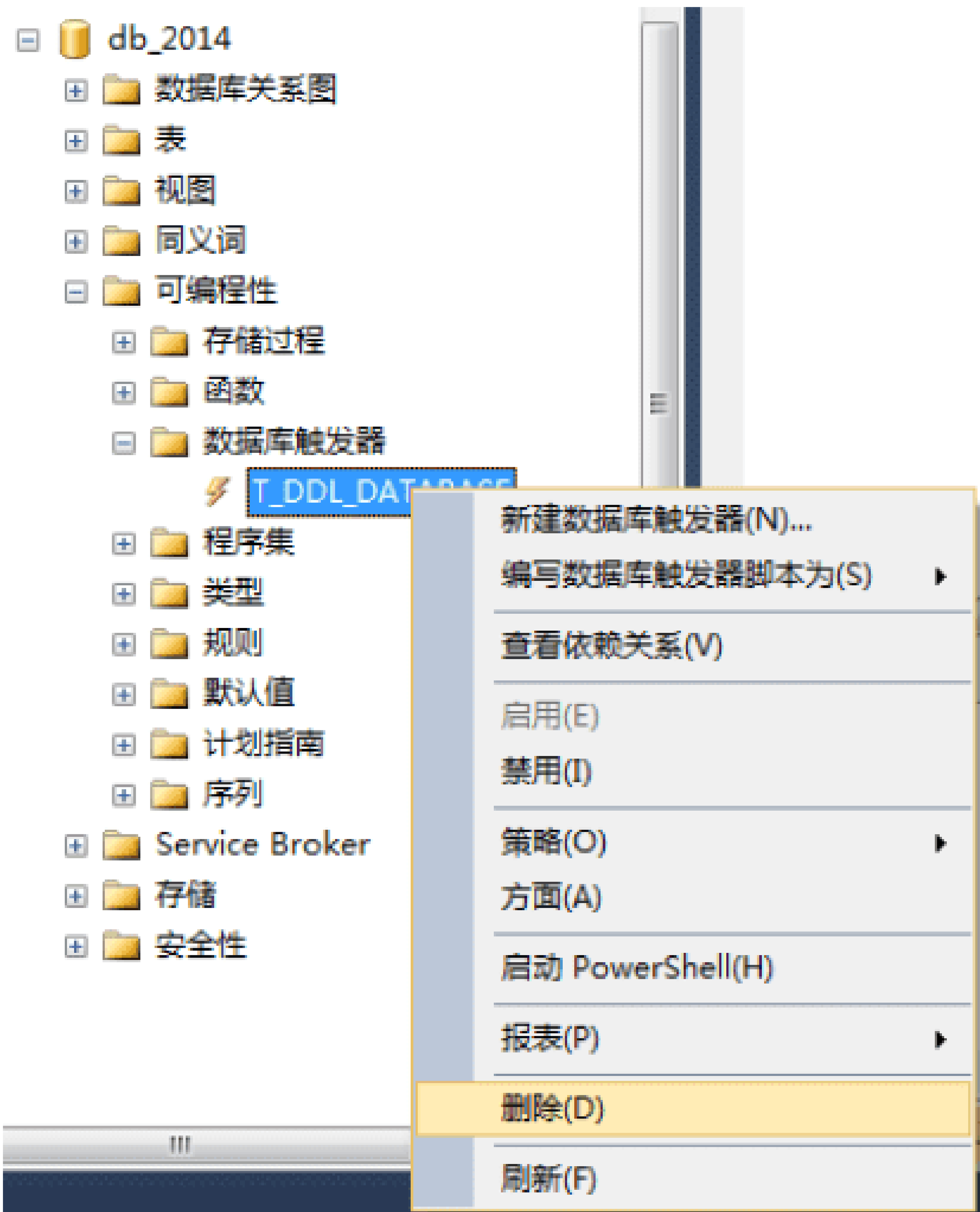


图 14.10　展开触发器所在的位置

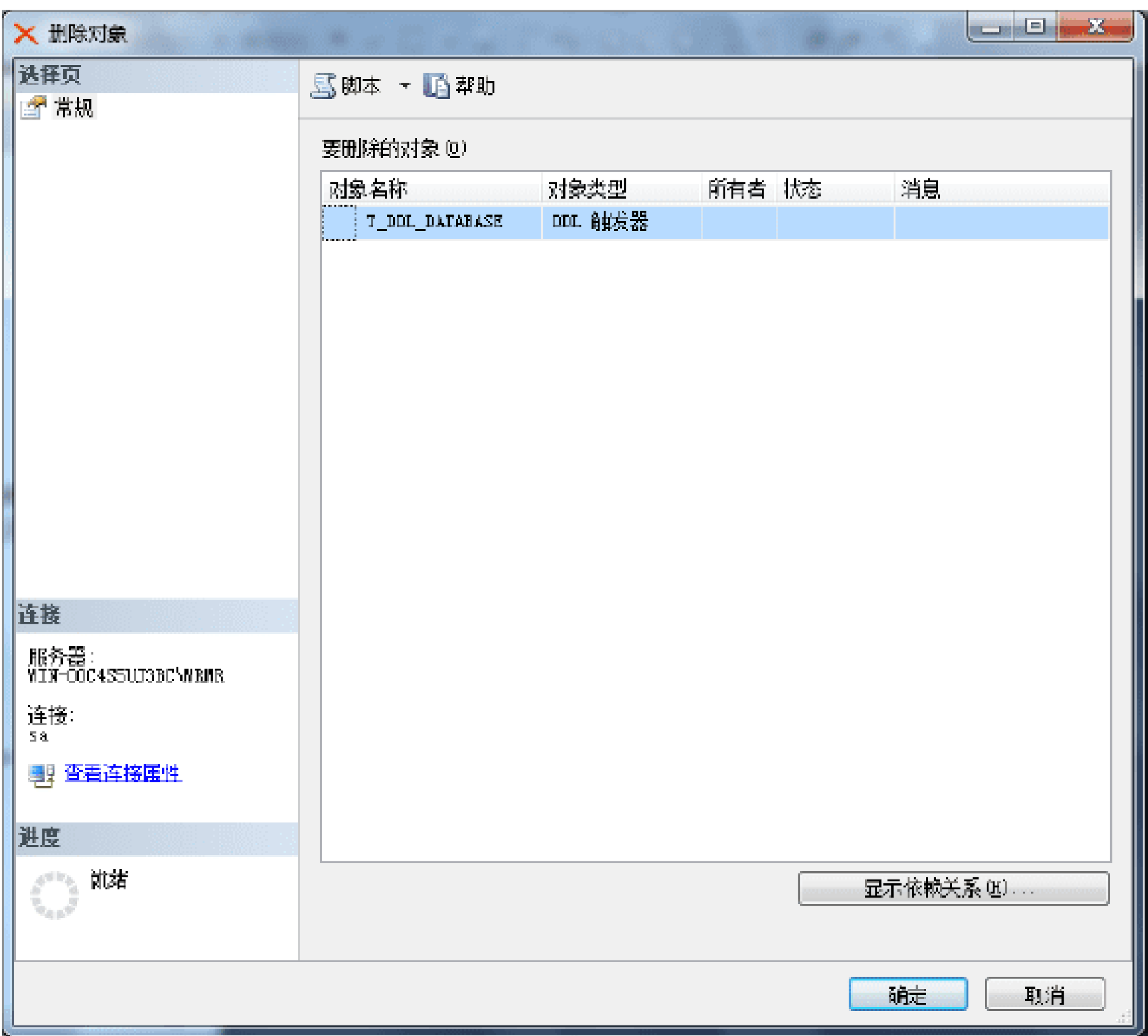


图 14.11　“删除对象”窗口

（4）在“删除对象”窗口中确认所删除的触发器，单击“确定”按钮即可将该触发器删除。

14.4　小　　结

本章介绍了触发器的概念，以及创建和管理触发器的方法。读者使用触发器可以在操作数据的同时触发指定的事件从而维护数据完整性。触发器可分为 DML 触发器、DDL 触发器和登录触发器，可以使用 SQL Server Management Studio 或者 Transact-SQL 语句对触发器进行管理。通过本章的学习，希望读者能更深入地熟悉触发器的使用。

第15章

游标的使用

（视频讲解：12分钟）

游标是取用一组数据并能够一次与一个单独的数据进行交互的方法，然而，不能通过在整个行集中修改或者选取数据来获得所需要的结果。本章将对游标的使用进行详细讲解。

学习摘要：

- **游标的概念**
- **游标的类型**
- **游标的基本操作**
- **游标系统存储过程**
- **使用系统过程查看游标的方法**

15.1　游标的概述



游标是取用一组数据并能够一次与一个单独的数据进行交互的方法。关系数据库中的操作会对整个行集起作用。由 SELECT 语句返回的行集包括满足该语句的 WHERE 子句中条件的所有行。这种由语句返回的完整行集称为结果集。应用程序，特别是交互式联机应用程序，并不总能将整个结果集作为一个单元来有效地处理。这些应用程序需要一种机制以便每次处理一行或一部分行。游标就是提供这种机制并对结果集的一种扩展。

游标通过以下方式来扩展结果处理。

☑　允许定位在结果集的特定行。

☑　从结果集的当前位置检索一行或一部分行。

☑　支持对结果集中当前位置的行进行数据修改。

☑　为由其他用户对显示在结果集中的数据库数据所做的更改提供不同级别的可见性支持。

☑　提供脚本、存储过程和触发器中用于访问结果集中的数据的 Transact-SQL 语句。

游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

15.1.1　游标的实现

游标提供了一种从表中检索数据并进行操作的灵活手段，游标主要用在服务器上，处理由客户端发送给服务器端的 SQL 语句，或是批处理、存储过程、触发器中的数据处理请求。游标的优点在于它可以定位到结果集中的某一行，并可以对该行数据执行特定操作，为用户在处理数据的过程中提供了很大方便。一个完整的游标由 5 部分组成，并且这 5 个部分应符合下面的顺序。

（1）声明游标。

（2）打开游标。

（3）从一个游标中查找信息。

（4）关闭游标。

（5）释放游标。

15.1.2　游标的类型

SQL Server 提供了 4 种类型的游标：静态游标、动态游标、只进游标和键集驱动游标。这些游标的检测结果集变化的能力和内存占用的情况都有所不同，数据源没有办法通知游标当前提取行的更改。游标检测这些变化的能力也受事务隔离级别的影响。

1．静态游标

静态游标的完整结果集在游标打开时建立在 tempdb 中。静态游标总是按照游标打开时的原样显示

结果集。静态游标在滚动期间很少或根本检测不到变化，虽然它在 tempdb 中存储了整个游标，但消耗的资源很少。尽管动态游标使用 tempdb 的程度最低，在滚动期间它能够检测到所有变化，但消耗的资源也更多。键集驱动游标介于二者之间，它能检测到大部分的变化，但比动态游标消耗更少的资源。

2．动态游标

动态游标与静态游标相对。当滚动游标时，动态游标反映结果集中所做的所有更改。结果集中的行数据值、顺序和成员在每次提取时都会改变。所有用户做的全部 UPDATE、INSERT 和 DELETE 语句均通过游标可见。

3．只进游标

只进游标不支持滚动，它只支持游标从头到尾顺序提取。只在从数据库中提取出来后才能进行检索。对所有由当前用户发出或由其他用户提交、并影响结果集中的行的 INSERT、UPDATE 和 DELETE 语句，其效果在这些行从游标中提取时是可见的。

4．键集驱动游标

打开游标时，键集驱动游标中的成员和行顺序是固定的。键集驱动游标由一套被称为键集的唯一标识符（键）控制。键由以唯一方式在结果集中标识行的列构成。键集是游标打开时来自所有适合 SELECT 语句的行中的一系列键值。键集驱动游标的键集在游标打开时建立在 tempdb 中。对非键集列中的数据值所做的更改（由游标所有者更改或其他用户提交）在用户滚动游标时是可见的。在游标外对数据库所做的插入在游标内是不可见的，除非关闭并重新打开游标。


视频讲解

15.2　游标的基本操作

游标的基本操作包括声明游标、打开游标、读取游标中的数据、关闭游标和释放游标。本节将详细地介绍如何操作游标。

15.2.1　声明游标

声明游标可以使用 DECLARE CURSOR 语句。此语句有两种语法声明格式，分别为 ISO 标准语法和 Transact-SQL 扩展的语法，下面将分别介绍声明游标的两种语法格式。

1．ISO 标准语法

语法格式如下：

```
DECLARE cursor_name [INSENSITIVE] [SCROLL] CURSOR
FOR select_statement
FOR {READ ONLY | UPDATE [OF column_name [,...n]]}
```

参数说明如下。

☑　DECLARE cursor_name：指定一个游标名称，其游标名称必须符合标识符规则。

☑ INSENSITIVE：定义一个游标，以创建将由该游标使用的数据的临时复本。对游标的所有请求都从 tempdb 中的临时表中得到应答；因此，在对该游标进行提取操作时返回的数据中不反映对基表所做的修改，并且该游标不允许修改。使用 SQL-92 语法时，如果省略 INSENSITIVE，（任何用户）对基表提交的删除和更新都反映在后面的提取中。

☑ SCROLL：指定所有的提取选项（FIRST、LAST、PRIOR、NEXT、RELATIVE、ABSOLUTE）均可用。

- FIRST：取第一行数据。
- LAST：取最后一行数据。
- PRIOR：取前一行数据。
- NEXT：取后一行数据。
- RELATIVE：按相对位置取数据。
- ABSOLUTE：按绝对位置取数据。

如果未指定 SCROLL，则 NEXT 是唯一支持的提取选项。

☑ select_statement：定义游标结果集的标准 SELECT 语句。在游标声明的 select_statement 内不允许使用关键字 COMPUTE、COMPUTE BY、FOR BROWSE 和 INTO。

☑ READ ONLY：表明不允许游标内的数据被更新，尽管在默认状态下游标是允许更新的。在 UPDATE 或 DELETE 语句的 WHERE CURRENT OF 子句中不允许引用游标。

☑ UPDATE [OF column_name [,...n]]：定义游标内可更新的列。如果指定 OF column_name [,...n] 参数，则只允许修改所列出的列。如果在 UPDATE 中未指定列的列表，则可以更新所有列。

2．Transact-SQL 扩展的语法

语法格式如下：

```
DECLARE cursor_name CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR select_statement
[FOR UPDATE [OF column_name [,...n]]]
```

DECLARE CURSOR 语句的参数及说明如表 15.1 所示。

表 15.1　DECLARE CURSOR 语句的参数及说明

参　　数	描　　述
DECLARE cursor_name	指定一个游标名称，其游标名称必须符合标识符规则
LOCAL	定义游标的作用域仅限在其所在的批处理、存储过程或触发器中。当建立游标在存储过程执行结束后，游标会被自动释放
GLOBAL	指定该游标的作用域对连接是全局的。在由连接执行的任何存储过程或批处理中，都可以引用该游标名称。该游标仅在脱接时隐性释放

续表

参　数	描　述
FORWARD_ONLY	指定游标只能从第一行滚动到最后一行。FETCH NEXT 是唯一受支持的提取选项非指定 STATIC、KEYSET 或 DYNAMIC 关键字，否则默认为 FORWARD_ONLY。STATIC、KEYSET 和 DYNAMIC 游标默认为 SCROLL。与 ODBC 和 ADO 这类数据库 API 不同，STATIC、KEYSET 和 DYNAMICTransact-SQL 游标支持 FORWARD_ONLY。FAST_FORWARD 和 FORWARD_ONLY 是互斥的；如果指定一个，则不能指定另一个
STATIC	定义一个游标，以创建将由该游标使用的数据的临时复本。对游标的所有请求都从 tempdb 中的该临时表中得到应答；因此，在对该游标进行提取操作时返回的数据中不反映对基表所做的修改，并且该游标不允许修改
KEYSET	指定当游标打开时，游标中行的成员资格和顺序已经固定。对行进行唯一标识的键集内置在 tempdb 内一个称为 keyset 的表中。对基表中的非键值所做的更改（由游标所有者更改或由其他用户提交）在用户滚动游标时是可视的。其他用户进行的插入是不可视的（不能通过 Transact-SQL 服务器游标进行插入）。如果某行已删除，则对该行的提取操作将返回 @@FETCH_STATUS 值-2。从游标外更新键值类似于删除旧行后接着插入新行的操作。含有新值的行不可视，对含有旧值的行的提取操作将返回@@FETCH_STATUS 值-2。如果通过指定 WHERE CURRENT OF 子句用游标完成更新，则新值可视
DYNAMIC	定义一个游标，以反映在滚动游标时对结果集内的行所做的所有数据的更改。行的数据值、顺序和成员在每次提取时都会更改。动态游标不支持 ABSOLUTE 提取选项
FAST_FORWARD	指明一个 FORWARD_ONLY、READ_ONLY 型游标
SCROLL_LOCKS	指定确保通过游标完成的定位更新或定位删除可以成功。将行读入游标以确保它们可用于以后的修改时，SQL Server 会锁定这些行。如果还指定了 FAST_FORWARD，则不能指定 SCROLL_LOCKS
OPTIMISTIC	指明在数据被读入游标后，如果游标中某行数据已发生变化，那么对游标数据进行更新或删除可能会导致失败
TYPE_WARNING	指定如果游标从所请求的类型隐性转换为另一种类型，则给客户端发送警告消息

【例 15.01】　创建一个名为 Cur_Emp 的标准游标。（**实例位置：资源包\源码\ 15\15.01**）

SQL 语句如下：

```
USE db_2014
DECLARE Cur_Emp CURSOR FOR
SELECT * FROM Employee
GO
```

运行结果如图 15.1 所示。

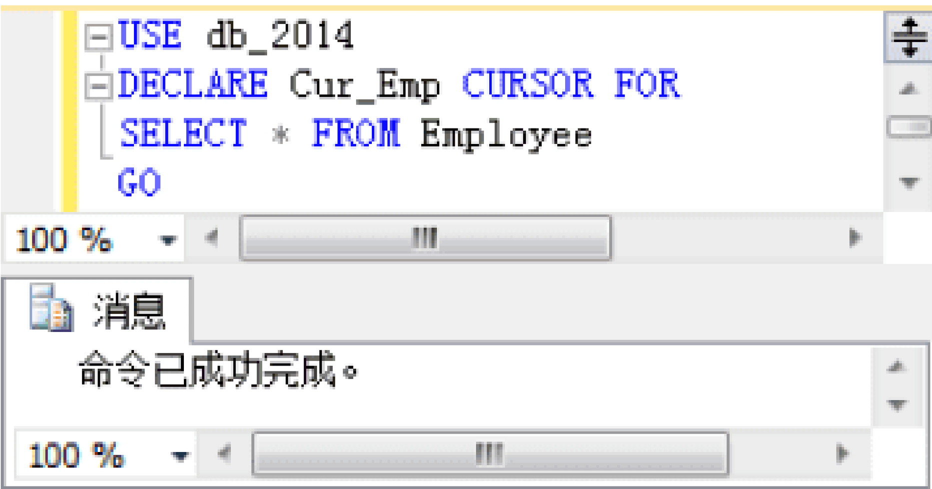


图 15.1　创建标准游标

【例 15.02】　创建一个名为 Cur_Emp_01 的只读游标。（**实例位置：资源包\源码\15\15.02**）

SQL 语句如下：

```
USE db_2014
DECLARE Cur_Emp_01 CURSOR FOR
SELECT * FROM Employee
FOR READ ONLY       --只读游标
GO
```

运行结果如图 15.2 所示。

【例 15.03】　创建一个名为 Cur_Emp_02 的更新游标。（**实例位置：资源包\源码\15\15.03**）

SQL 语句如下：

```
USE db_2014
DECLARE Cur_Emp_02 CURSOR FOR
SELECT Name,Sex,Age FROM Employee
FOR UPDATE       --更新游标
GO
```

运行结果如图 15.3 所示。

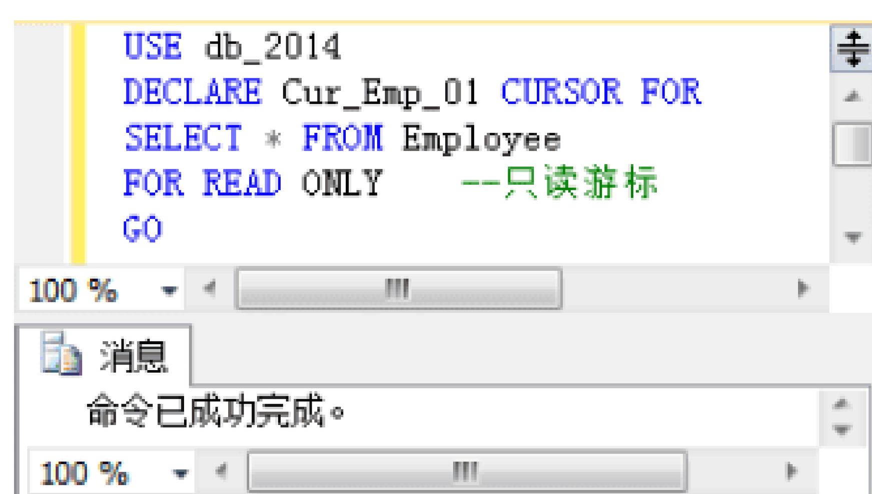


图 15.2　创建只读游标

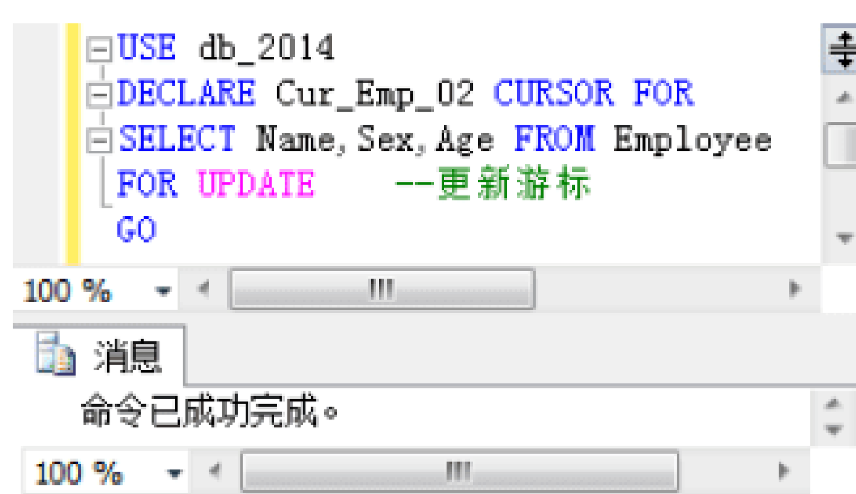


图 15.3　创建更新游标

15.2.2　打开游标

打开一个声明的游标可以使用 OPEN 命令。语法格式如下：

```
OPEN {{[GLOBAL] cursor_name} | cursor_variable_name}
```

参数说明如下。

☑ GLOBAL：指定 cursor_name 为全局游标。

☑ cursor_name：已声明的游标名称，如果全局游标和局部游标都使用 cursor_name 作为其名称，那么如果指定了 GLOBAL，cursor_name 指的是全局游标，否则，cursor_name 指的是局部游标。

☑ cursor_variable_name：游标变量的名称，该名称引用一个游标。



说明

如果使用 INSENSITIV 或 STATIC 选项声明了游标，那么 OPEN 将创建一个临时表以保留结果集。如果结果集中任意行的大小超过 SQL Server 表的最大行大小，OPEN 将失败。如果使用 KEYSET 选项声明了游标，那么 OPEN 将创建一个临时表以保留键集。临时表存储在 tempdb 中。

【例 15.04】 首先声明一个名为 Emp_01 的游标，然后使用 OPEN 命令打开该游标。（**实例位置：资源包\源码\15\15.04**）

SQL 语句如下：

```
USE db_2014
DECLARE Emp_01 CURSOR FOR          --声明游标
SELECT * FROM Employee
WHERE ID = '1'
OPEN Emp_01                        --打开游标
GO
```

运行结果如图 15.4 所示。

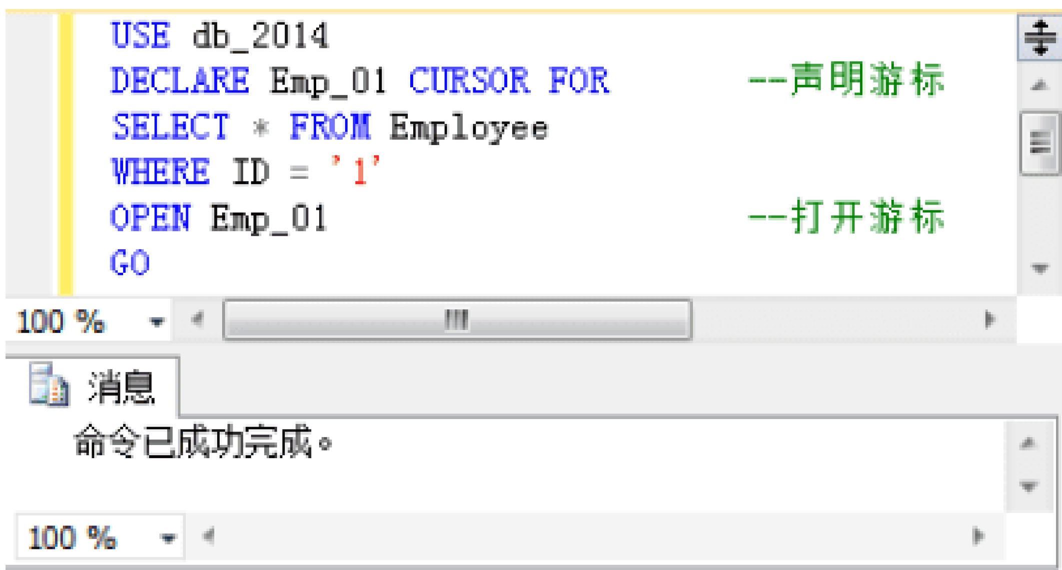


图 15.4 打开游标

15.2.3 读取游标中的数据

当打开一个游标之后，就可以读取游标中的数据了。可以使用 FETCH 命令读取游标中的某一行数据。语法格式如下：

```
FETCH
        [[NEXT | PRIOR | FIRST | LAST
              | ABSOLUTE {n | @nvar}
              | RELATIVE {n | @nvar}
         ]
          FROM
        ]
{{[GLOBAL] cursor_name} | @cursor_variable_name}
[INTO @variable_name [,...n]]
```

FETCH 命令的参数及说明如表 15.2 所示。

表 15.2 FETCH 命令的参数及说明

参　　数	描　　述
NEXT	返回紧跟当前行之后的结果行，并且当前行递增为结果行。如果 FETCH NEXT 为对游标的第一次提取操作，则返回结果集中的第一行。NEXT 为默认的游标提取选项
PRIOR	返回紧临当前行前面的结果行，并且当前行递减为结果行。如果 FETCH PRIOR 为对游标的第一次提取操作，则没有行返回并且游标置于第一行之前
FIRST	返回游标中的第一行并将其作为当前行

续表

参　　数	描　　述
LAST	返回游标中的最后一行并将其作为当前行
ABSOLUTE {n | @nvar}	如果 n 或@nvar 为正数，返回从游标头开始的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为负数，返回游标尾之前的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为 0，则没有行返回
RELATIVE {n | @nvar}	如果 n 或@nvar 为正数，返回当前行之后的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为负数，返回当前行之前的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为 0，返回当前行。如果对游标的第一次提取操作时将 FETCHRELATIVE 的 n 或@nvar 指定为负数或 0，则没有行返回。n 必须为整型常量且@nvar 必须为 smallint、tinyint 或 int
GLOBAL	指定 cursor_name 为全局游标
cursor_name	要从中进行提取的开放游标的名称。如果同时有以 cursor_name 作为名称的全局和局部游标存在，若指定为 GLOBAL，则 cursor_name 对应于全局游标，未指定 GLOBAL，则对应于局部游标
@cursor_variable_name	游标变量名，引用要进行提取操作的打开的游标
INTO @variable_name[,...n]	允许将提取操作的列数据放到局部变量中。列表中的各个变量从左到右与游标结果集中的相应列相关联。各变量的数据类型必须与相应的结果列的数据类型匹配，或是结果列数据类型所支持的隐性转换。变量的数目必须与游标选择列表中的列的数目一致
@@FETCH_STATUS	返回上次执行 FETCH 命令的状态。在每次用 FETCH 从游标中读取数据时，都应检查该变量，以确定上次 FETCH 操作是否成功，决定如何进行下一步处理。@@FETCH_STATUS 变量有 3 个不同的返回值，说明如下：（1）返回值为 0：FETCH 语句成功；（2）返回值为-1：FETCH 语句失败或此行不在结果集中；（3）返回值为-2：被提取的行不存在


说明

（1）在前两个参数中，包含了 n 和@nvar，表示游标相对于作为基准的数据行所偏离的位置。

（2）当使用 SQL-92 语法来声明一个游标时，没有选择 SCROLL 选项，则只能使用 FETCH NEXT 命令来从游标中读取数据，即只能从结果集第一行按顺序地每次读取一行。由于不能使用 FIRST、LAST、PRIOR，所以无法回滚读取以前的数据。如果选择了 SCROLL 选项，则可以使用所有的 FETCH 操作。

【例 15.05】 用@@FETCH_STATUS 控制一个 WHILE 循环中的游标活动，SQL 语句及运结果如图 15.5 所示。（**实例位置：资源包\源码\15\15.05**）

SQL 语句如下：

```
USE db_2014                          --引入数据库
DECLARE ReadCursor CURSOR FOR        --声明一个游标
SELECT * FROM Student
OPEN ReadCursor                      --打开游标
FETCH NEXT FROM ReadCursor           --执行取数操作
WHILE @@FETCH_STATUS=0               --检查@@FETCH_STATUS，以确定是否还可以继续取数
```

```
BEGIN
  FETCH NEXT FROM ReadCursor
END
```

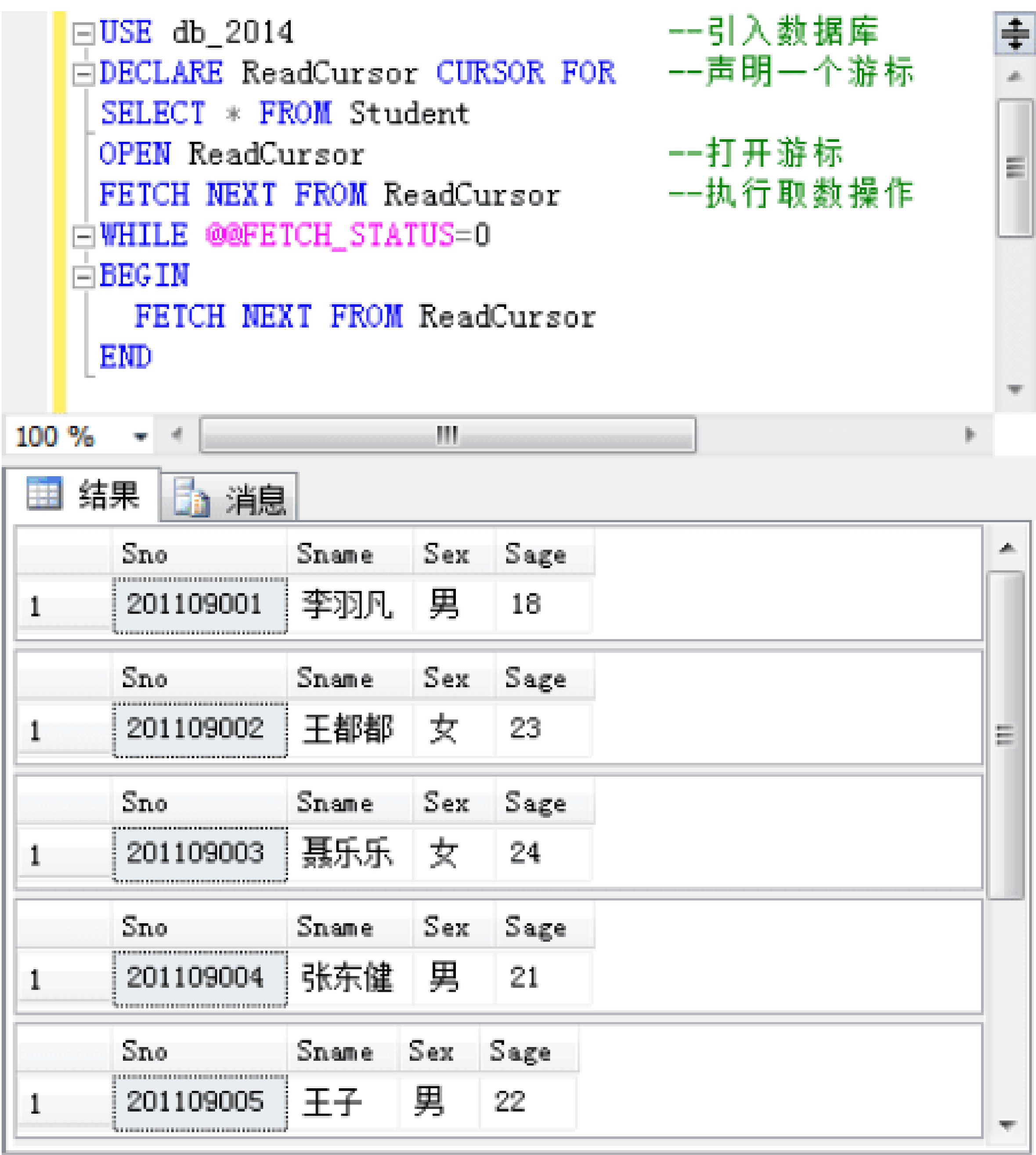


图 15.5　从游标中读取数据

15.2.4　关闭游标

当游标使用完毕之后，使用 CLOSE 语句可以关闭游标，但不释放游标占用的系统资源。语法格式如下：

```
CLOSE {{[GLOBAL] cursor_name} | cursor_variable_name}
```

参数说明如下。

☑　GLOBAL：指定 cursor_name 为全局游标。

☑　cursor_name：开放游标的名称。如果全局游标和局部游标都使用 cursor_name 作为它们的名称，那么当指定 GLOBAL 时，cursor_name 引用全局游标；否则，cursor_name 引用局部游标。

☑　cursor_variable_name：与开放游标关联的游标变量名称。

【例 15.06】　声明一个名为 CloseCursor 的游标，并使用 Close 语句关闭游标。（**实例位置：资源包\源码\15\15.06**）

SQL 语句如下：

```
USE db_2014
DECLARE CloseCursor Cursor FOR
SELECT * FROM   Student
FOR READ ONLY
OPEN CloseCursor
CLOSE CloseCursor
```

运行结果如图 15.6 所示。

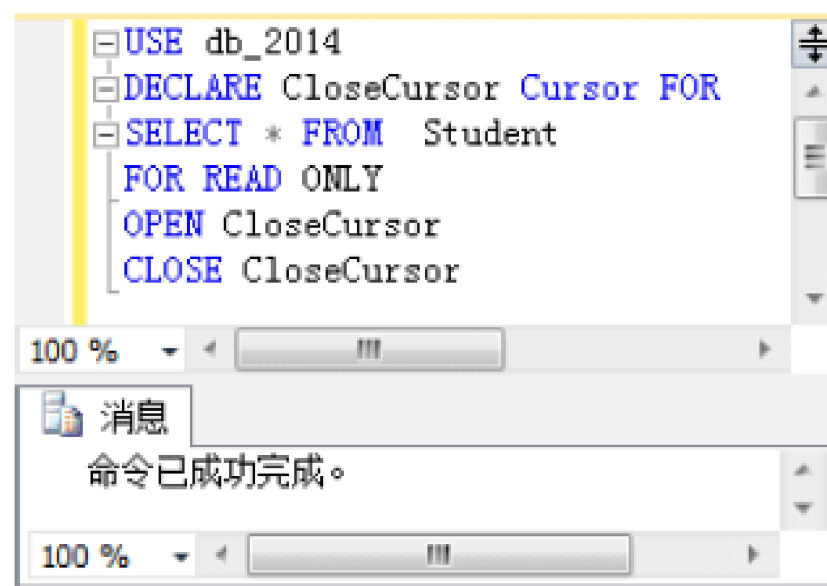


图 15.6　关闭游标

15.2.5　释放游标

当游标关闭之后，并没有在内存中释放所占用的系统资源，所以可以使用 DEALLOCATE 命令删除游标引用。当释放最后的游标引用时，组成该游标的数据结构由 SQL Server 释放。语法格式如下：

```
DEALLOCATE {{[GLOBAL] cursor_name} | @cursor_variable_name}
```

参数说明如下。

☑ cursor_name：已声明游标的名称。当全局和局部游标都以 cursor_name 作为它们的名称存在时，如果指定 GLOBAL，则 cursor_name 引用全局游标，如果未指定 GLOBAL，则 cursor_name 引用局部游标。

☑ @cursor_variable_name：cursor 变量的名称。@cursor_variable_name 必须为 cursor 类型。

当使用 DEALLOCATE @cursor_variable_name 来删除游标时，游标变量并不会被释放，除非超过使用该游标的存储过程和触发器的范围。

【例 15.07】 使用 DEALLOCATE 命令释放名为 FreeCursor 的游标。（**实例位置：资源包\源码\15\15.07**）

SQL 语句如下：

```
USE db_2014
DECLARE FreeCursor Cursor FOR
SELECT * FROM Student
OPEN FreeCursor
Close FreeCursor
DEALLOCATE FreeCursor
```

运行结果如图 15.7 所示。

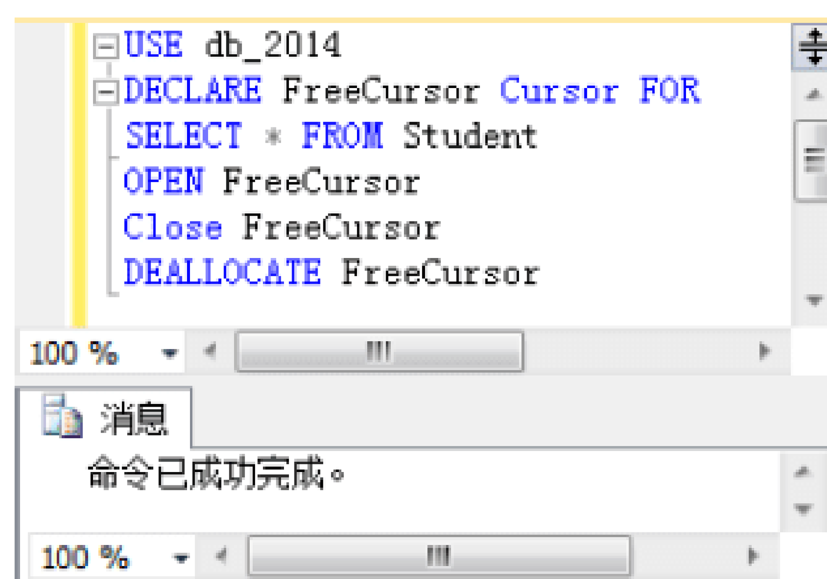


图 15.7　释放游标


视频讲解

15.3　使用系统过程查看游标

创建游标后，通常使用 sp_cursor_list 和 sp_describe_cursor 查看游标的属性。sp_cursor_list 用来报告当前为连接打开的服务器游标的属性，sp_describe_cursor 用于报告服务器游标的属性。本节将详细地介绍这两个系统过程。

15.3.1　sp_cursor_list

sp_cursor_list 报告当前为连接打开的服务器游标的属性。语法格式如下：

```
sp_cursor_list [@cursor_return =] cursor_variable_name OUTPUT
          , [@cursor_scope =] cursor_scope
```

参数说明如下。

☑ [@cursor_return =] cursor_variable_name OUTPUT：已声明的游标变量的名称。cursor_ variable_name 的数据类型为 cursor，无默认值。游标是只读的可滚动动态游标。

☑ [@cursor_scope =] cursor_scope：指定要报告的游标级别。cursor_scope 的数据类型为 int，无默认值，可取值如表 15.3 所示。

表 15.3　cursor_scope 可取的值

值	说　明
1	报告所有本地游标
2	报告所有全局游标
3	报告本地游标和全局游标

【例 15.08】　声明一个游标 Cur_Employee，并使用 sp_cursor_list 报告该游标的属性。（**实例位置：资源包\源码\15\15.08**）

SQL 语句如下：

```
USE db_2014
GO
DECLARE Cur_Employee CURSOR FOR
SELECT Name
FROM Employee
WHERE Name LIKE '王%'
OPEN Cur_Employee
DECLARE @Report CURSOR
EXEC master.dbo.sp_cursor_list @cursor_return = @Report OUTPUT,
        @cursor_scope = 2
FETCH NEXT from @Report
WHILE (@@FETCH_STATUS <> -1)
```

```
BEGIN
    FETCH NEXT from @Report
END
CLOSE @Report
DEALLOCATE @Report
GO
CLOSE Cur_Employee
DEALLOCATE Cur_Employee
GO
```

运行结果如图 15.8 所示。

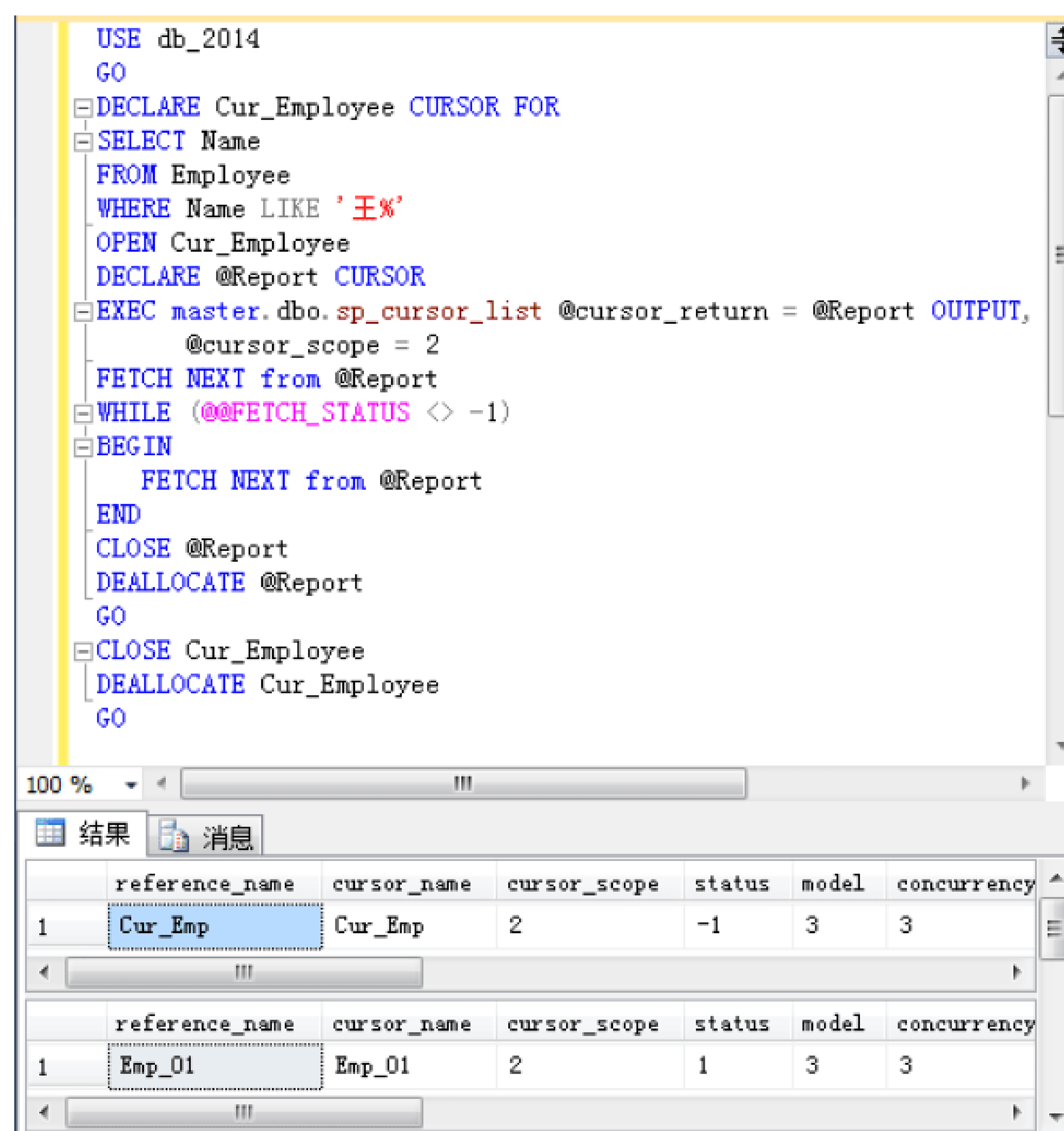


图 15.8　sp_cursor_list 属性

15.3.2　sp_describe_cursor

sp_describe_cursor 用于报告服务器游标的属性。语法格式如下：

```
sp_describe_cursor [@cursor_return =] output_cursor_variable OUTPUT
        {[, [@cursor_source =] N'local'
        , [@cursor_identity =] N'local_cursor_name']
        | [, [@cursor_source =] N'global'
        , [@cursor_identity =] N'global_cursor_name']
        | [, [@cursor_source =] N'variable'
        , [@cursor_identity =] N'input_cursor_variable']
        }
```

sp_describe_cursor 语句的参数及说明如表 15.4 所示。

表 15.4　sp_describe_cursor 语句的参数及说明

参　　数	描　　述
[@cursor_return =] output_cursor_variable OUTPUT	用于接收游标输出的声明游标变量的名称。output_cursor_variable 的数据类型为 cursor，无默认值。调用 sp_describe_ cursor 时，该参数不得与任何游标关联。返回的游标是可滚动的动态只读游标
[@cursor_source =] {N'local'| N'global' | N'variable'}	指定是使用局部游标的名称、全局游标的名称还是游标变量的名称来指定要报告的游标。该参数的类型为 nvarchar(30)
[@cursor_identity =] N'local_cursor_name'	由具有 LOCAL 关键字或默认设置为 LOCAL 的 DECLARE CURSOR 语句创建的游标名称。local_cursor_name 的数据类型为 nvarchar(128)
[@cursor_identity =] N'global_cursor_name'	由具有 GLOBAL 关键字或默认设置为 GLOBAL 的 DECLARE CURSOR 语句创建的游标名称。global_cursor_name 的数据类型为 nvarchar(128)
[@cursor_identity =] N'input_cursor_variable'	与所打开游标相关联的游标变量的名称。input_cursor_variable 的数据类型为 nvarchar(128)

【例 15.09】　声明一个游标，并使用 sp_describe_cursor 报告该游标的属性。（**实例位置：资源包\源码\15\15.09**）

SQL 语句如下：

```
USE db_2014
GO
DECLARE Cur_Employee CURSOR STATIC FOR
SELECT Name
FROM Employee
OPEN Cur_Employee
DECLARE @Report CURSOR
EXEC master.dbo.sp_describe_cursor @cursor_return = @Report OUTPUT,
        @cursor_source = N'global', @cursor_identity = N'Cur_Employee'
FETCH NEXT from @Report
WHILE (@@FETCH_STATUS <> -1)
BEGIN
    FETCH NEXT from @Report
END
CLOSE @Report
DEALLOCATE @Report
GO
CLOSE Cur_Employee
DEALLOCATE Cur_Employee
GO
```

运行结果如图 15.9 所示。

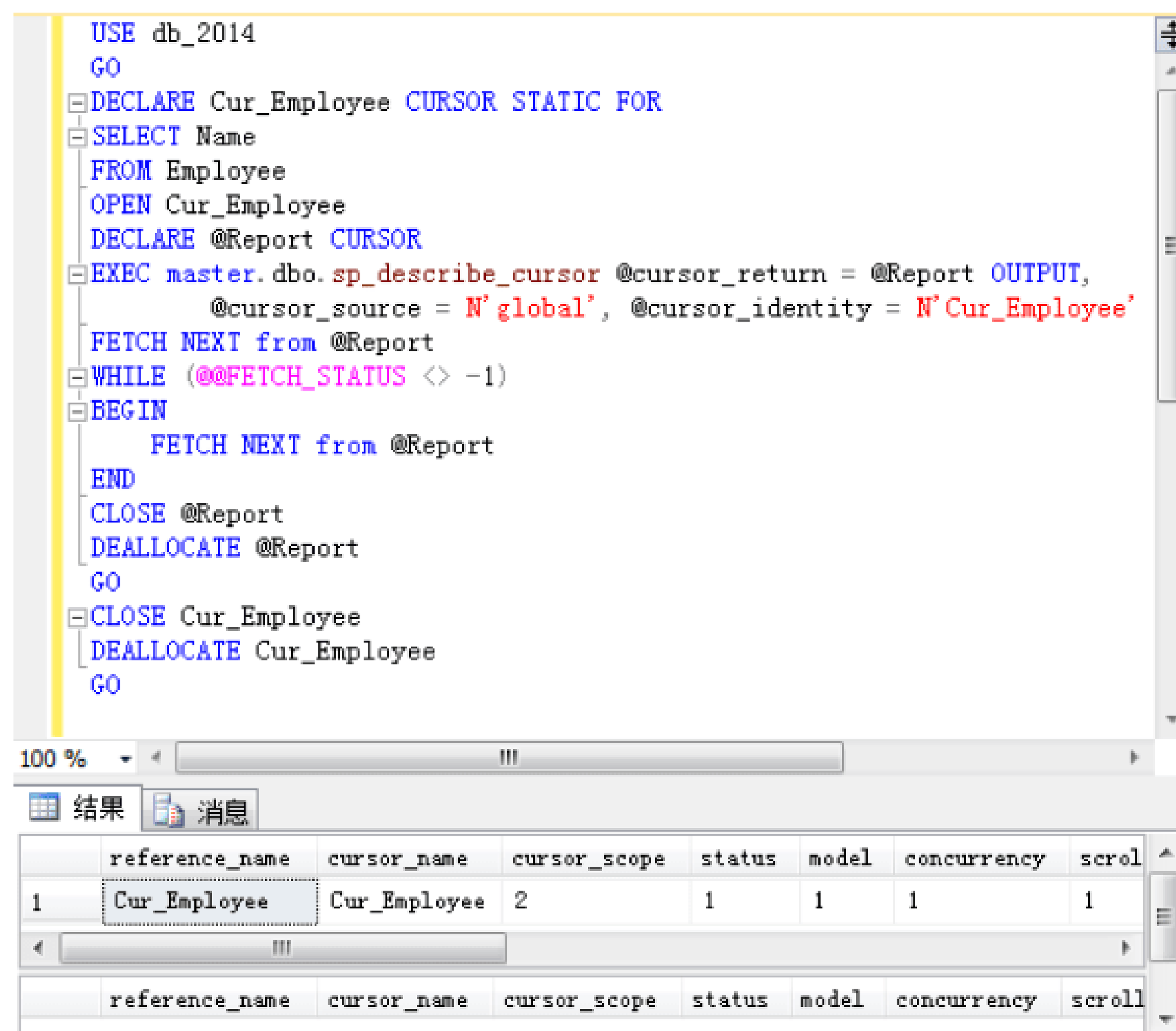


图 15.9　sp_describe_cursor 属性

15.4　小　　结

本章主要介绍了游标的概念、类型及游标的基本操作。游标为应用程序提供了每次对结果集处理一行或一部分行的机制。虽然游标可以解决结果集无法完成的所有操作，但要避免使用游标，游标非常消耗资源，而且会对性能产生很大的影响。游标只能在别无选择的时候使用。

第16章

SQL 中的事务

（视频讲解：28 分钟）

在数据提交过程中，事务非常重要，它是一个独立的工作单元，如果某一事务成功，则在该事务中进行的所有数据修改均会提交，成为数据库中的永久组成部分，如果事务遇到错误且必须取消或回滚，则所有数据修改均被清除。本章将从事务的概念、显示与隐式事务、使用事务、事务的工作机制、事务的并发、锁和分布式事务处理等多个方面对SQL中的事务进行详细讲解。

学习摘要：

- 事务的概念
- 显式事务与隐式事务
- 使用事务
- 事务的工作机制
- 自动提交事务
- 事务的并发问题
- 事务的隔离级别
- 锁的机制
- 死锁的产生原理
- 分布式事务处理

16.1　事务的概念



事务是由一系列语句构成的逻辑工作单元。事务和存储过程等批处理有一定程度上的相似之处，通常都是为了完成一定业务逻辑而将一条或者多条语句“封装”起来，使它们与其他语句之间出现一个逻辑上的边界，并形成相对独立的一个工作单元。

当使用事务修改多个数据表时，如果在处理的过程中出现了某种错误，如系统死机或突然断电等情况，则返回结果是数据全部没有被保存。因为事务处理的结果只有两种：一种是在事务处理的过程中，如果发生了某种错误则整个事务全部回滚，使所有对数据的修改全部撤销，事务对数据库的操作是单步执行的，当遇到错误时可以随时回滚；另一种是如果没有发生任何错误且每一步的执行都成功，则整个事务全部被提交。从而可以看出，有效地使用事务不但可以提高数据的安全性，而且还可以增强数据的处理效率。

事务包含 4 种重要的属性，被统称为 ACID（原子性、一致性、隔离性和持久性），一个事务必须通过 ACID。

（1）原子性（Atomic）：事务是一个整体的工作单元，事务对数据库所做的操作要么全部执行，要么全部取消。如果某条语句执行失败，则所有语句全部回滚。

（2）一致性（ConDemoltent）：事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。如果事务成功，则所有数据将变为一个新的状态；如果事务失败，则所有数据将处于开始之前的状态。

（3）隔离性（Isolated）：由事务所做的修改必须与其他事务所做的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。

（4）持久性（Durability）：当事务提交后，对数据库所做的修改就会永久保存下来。

16.2　显式事务与隐式事务



事务是单个的工作单元。如果某一事务成功，则在该事务中进行的所有数据修改均会提交，成为数据库中的永久组成部分。如果事务遇到错误且必须取消或回滚，则所有数据修改均被清除。

SQL Server 以下列事务模式运行。

☑ 自动提交事务：每条单独的语句都是一个事务。

☑ 显式事务：每个事务均以 BEGIN TRANSACTION 语句显式开始，以 COMMIT 或 ROLLBACK 语句显式结束。

☑ 隐式事务：在前一个事务完成时新事务隐式启动，但每个事务仍以 COMMIT 或 ROLLBACK 语句显式完成。

☑ 批处理级事务：只能应用于多个活动结果集（MARS），在 MARS 会话中启动的 Transact-SQL

显式或隐式事务变为批处理级事务。当批处理完成时没有提交或回滚的批处理级事务自动由 SQL Server 进行回滚。

在本节中主要介绍显式事务和隐式事务。

16.2.1 显式事务

显式事务是用户自定义或用户指定的事务。可以通过 BEGIN TRANSACTION、COMMIT TRANSACTION、COMMIT WORK、ROLLBACK TRANSACTION 或 ROLLBACK WORK 事务处理语句定义显式事务。下面将简单介绍以上几种事务处理语句的语法和参数。

（1）BEGIN TRANSACTION 语句

用于启动一个事务，它标志着事务的开始。语法格式如下：

```
BEGIN TRAN [SACTION] [transaction_name | @tran_name_variable[WITH MARK ['description']]]
```

参数说明如下。

☑ transaction_name：表示设定事务的名称，字符个数最多为 32 个字符。

☑ @tran_name_variable：表示用户定义的、含有有效事务名称的变量名称，必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。

☑ WITH MARK ['description']：表示指定在日志中标记事务，description 是描述该标记的字符串。

（2）COMMIT TRANSACTION 语句

用于标志一个成功的隐式事务或用户定义事务的结束。语法格式如下：

```
COMMIT [TRAN [SACTION] [transaction_name | @tran_name_variable]]
```

参数说明如下。

☑ transaction_name：表示此参数指定由前面的 BEGIN TRANSACTION 指派的事务名称，此处的事务名称仅用来帮助程序员阅读，以及指明 COMMIT TRANSACTION 与哪些嵌套的 BEGIN TRANSACTION 相关联。

☑ @tran_name_variable：表示用户定义的、含有有效事务名称的变量名称，必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。


说明

如果@@TRANCOUNT 为 1，COMMIT TRANSACTION 使得自从事务开始以来所执行的所有数据修改成为数据库的永久部分，释放连接占用的资源，并将@@TRANCOUNT 减少到 0。如果@@TRANCOUNT 大于 1，则 COMMIT TRANSACTION 使@@TRANCOUNT 按 1 递减。

（3）COMMIT WORK 语句

用于标志事务的结束。语法格式如下：

```
COMMIT [WORK]
```

此语句的功能与 COMMIT TRANSACTION 相同，但 COMMIT TRANSACTION 接受用户定义的事

务名称。

（4）ROLLBACK TRANSACTION 语句

用于将显式事务或隐式事务回滚到事务的起点或事务内的某个保存点。当执行事务的过程中发生某种错误，可以使用 ROLLBACK TRANSACTION 语句或 ROLLBACK WORK 语句，使数据库撤销在事务中所做的更改，并使数据恢复到事务开始之前的状态。语法格式如下：

```
ROLLBACK [TRAN [SACTION] [transaction_name | @tran_name_variable| savepoint_name | @savepoint_variable]]
```

参数说明如下。

☑ transaction_name：表示 BEGIN TRAN 对事务名称的指派。
☑ @tran_name_variable：表示用户定义的、含有有效事务名称的变量名称，必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。
☑ savepoint_name：是来自 SAVE TRANSACTION 语句对保存点的定义，当条件回滚只影响事务的一部分时使用 savepoint_name。
☑ @savepoint_variable：表示用户定义的、含有有效保存点名称的变量名称。

（5）ROLLBACK WORK 语句

用于将用户定义的事务回滚到事务的起点。语法格式如下：

```
ROLLBACK [WORK]
```

此语句的功能与 ROLLBACK TRANSACTION 相同，除非 ROLLBACK TRANSACTION 接受用户定义的事务名称。

16.2.2　隐式事务

隐式事务需要使用 SET IMPLICIT_TRANSACTIONS ON 语句将隐式事务模式设置为打开。在打开了隐式事务的设置开关时，执行下一条语句时自动启动一个新事务，并且每关闭一个事务时，执行下一条语句又会启动一个新事务，直到关闭了隐式事务的设置开关。

SQL Server 的任何数据修改语句都是隐式事务，如 ALTER TABLE、CREATE、DELETE、DROP、FETCH、GRANT、INSERT、OPEN、REVOKE、SELECT、TRUNCATE TABLE、UPDATE。这些语句都可以作为一个隐式事务的开始。如果要结束隐式事务，需要使用 COMMIT TRANSACTION 或 ROLLBACK TRANSACTION 语句来结束事务。

16.2.3　API 中控制隐式事务

用来设置隐式事务的 API 机制是 ODBC 和 OLE DB。

（1）ODBC

☑ 调用 SQLSetConnectAttr 函数启动隐式事务模式，其中 Attribute 设置为 SQL_ATTR_AUTOCOMMIT，ValuePtr 设置为 SQL_AUTOCOMMIT_OFF。

- ☑ 在调用 SQLSetConnectAttr 之前，连接将一直保持为隐式事务模式，其中 Attribute 设置为 SQL_ATTR_AUTOCOMMIT，ValuePtr 设置为 SQL_AUTOCOMMIT_ON。
- ☑ 调用 SQLEndTran 函数提交或回滚每个事务，其中 CompletionType 设置为 SQL_COMMIT 或 SQL_ROLLBACK。

（2）OLE DB

OLE DB 没有专门用来设置隐式事务模式的方法。

- ☑ 调用 ITransactionLocal::StartTransaction 方法启动显式模式。
- ☑ 当调用 ITransaction::Commit 或 ITransaction::Abort 方法（其中，fRetaining 设置为 TRUE）时，OLE DB 将完成当前的事务并进入隐式事务模式。只要 ITransaction::Commit 或 ITransaction::Abort 中的 fRetaining 设置为 TRUE，那么连接就将保持隐式事务模式。
- ☑ 调用 ITransaction::Commit 或 ITransaction::Abort（其中 fRetaining 设置为 FALSE）停止隐式事务模式。

16.2.4　事务的 COMMIT 和 ROLLBACK

结束事务包括“成功时提交事务”和“失败时回滚事务”两种情况，在 Transact-SQL 中可以使用 COMMIT 和 ROLLBACK 结束事务。

（1）COMMIT

提交事务，用在事务执行成功的情况下。COMMIT 语句保证事务的所有修改都被保存，同时 COMMIT 语句也释放事务中使用的资源，如事务使用的锁。

（2）ROLLBACK

回滚事务，用于事务在执行失败的情况下，将显式事务或隐式事务回滚到事务的起点或事务内的某个保存点。



16.3　使用事务

在掌握事务的概念与运行模式之后，本节继续介绍如何使用事务。

16.3.1　开始事务

当一个数据库连接启动事务时，在该连接上执行的所有 Transact-SQL 语句都是事务的一部分，直到事务结束。开始事务使用 BEGIN TRANSACTION 语句。下面将以示例的形式演示如何在 SQL 中使用开始事务。

【例 16.01】 使用事务修改 Employee 表中的数据，首先使用 BEGIN TRANSACTION 语句启动事务 update_data，然后修改指定条件的数据，最后使用 COMMIT TRANSACTION 提交事务，SQL 语句及运行结果如图 16.1 所示。（**实例位置：资源包\源码\16\16.01**）

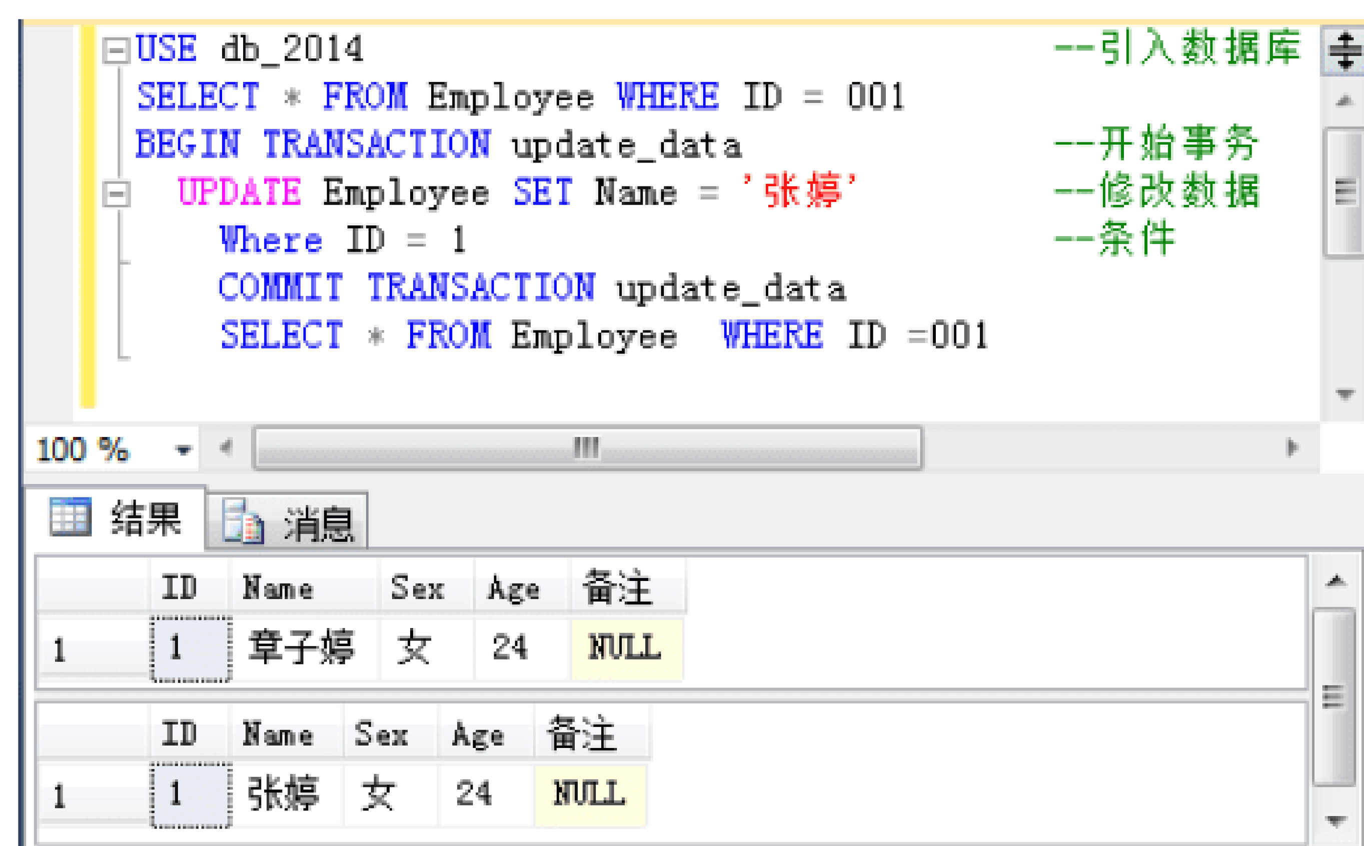


图 16.1　使用事务修改 Employee 表中的数据

SQL 语句如下：

```
USE db_2014                                    --引入数据库
SELECT * FROM Employee WHERE ID = 001
BEGIN TRANSACTION update_data                  --开始事务
  UPDATE Employee SET Name = '张婷'             --修改数据
     Where ID = 1                              --条件
     COMMIT TRANSACTION update_data
     SELECT * FROM Employee WHERE ID =001
```

在例 16.01 中，BEGIN TRANSACTION 语句指定一个事务的开始，update_data 语句为事务名称，它可由用户自定义，但必须是有效的标识符。COMMIT TRANSACTION 语句指定事务的结束。

**说明**

BEGIN TRANSACTION 与 COMMIT TRANSACTION 之间的语句，可以是任何对数据库进行修改的语句。

16.3.2　结束事务

当一个事务执行完成之后，要将其结束，以便释放所占用的内存资源，结束事务使用 COMMIT 语句。

【例 16.02】　使用事务在 Employee 表中添加一条记录，并使用 COMMIT 语句结束事务，SQL 语句及运行结果如图 16.2 所示。（**实例位置：资源包\源码\16\16.02**）

SQL 语句如下：

```
USE db_2014                                    --打开数据局
SELECT * FROM Employee
BEGIN TRANSACTION INSERT_DATA                  --开始事务
  INSERT INTO Employee
  VALUES('16','门闻双','女','22',NULL)
COMMIT TRANSACTION INSERT_DATA                 --结束事务
GO
IF @@ERROR = 0
```

```
  PRINT '插入新记录成功！'                    --输出插入成功的信息
GO
```

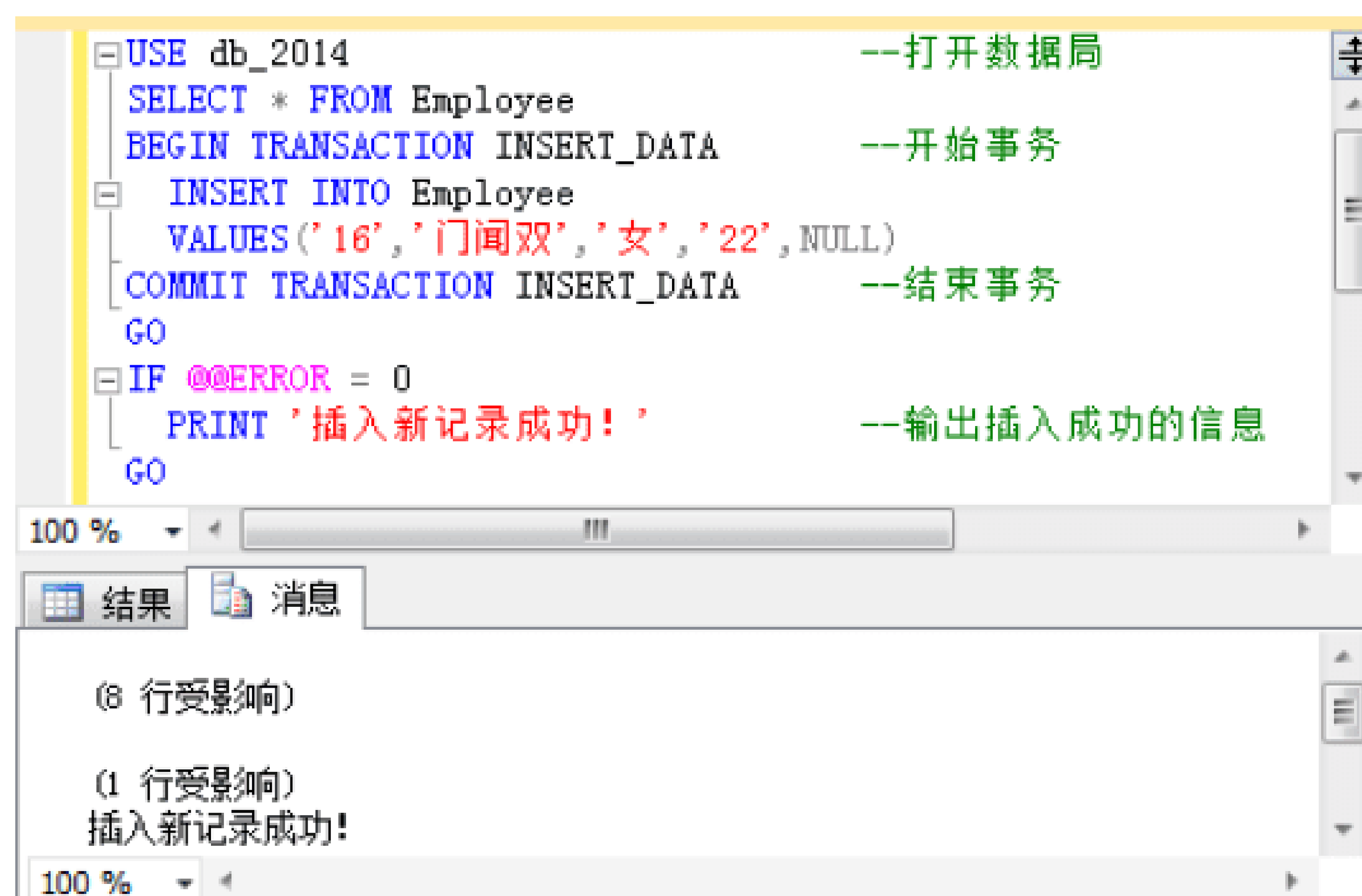


图 16.2　使用 COMMIT 结束事务

在例 16.02 中，使用了@@ERROR 函数，此函数用于判断最后的 Transact-SQL 语句是否执行成功。此函数有两个返回值，如果此语句执行成功，则@@ERROR 返回 0；如果此语句产生错误，则@@ERROR 返回错误号。每一个 Transact-SQL 语句完成时，@@ERROR 的值都会改变。

16.3.3　回滚事务

使用 ROLLBACK TRANSACTION 语句可以将显式事务或隐式事务回滚到事务的起点或事务内的某个保存点。语法格式如下：

```
ROLLBACK {TRAN | TRANSACTION}
     [transaction_name | @tran_name_variable
     | savepoint_name | @savepoint_variable]
[;]
```

参数说明如下。

- ☑ transaction_name：是为 BEGIN TRANSACTION 上的事务分配的名称（即事务名称），它必须符合标识符规则，但只使用事务名称的前 32 个字符，当嵌套事务时，transaction_name 必须是最外面的 BEGIN TRANSACTION 语句中的名称。
- ☑ @tran_name_variable：是用户定义的、包含有效事务名称的变量的名称，它必须用 char、varchar、nchar 或 nvarchar 数据类型声明变量。
- ☑ savepoint_name：是 SAVE TRANSACTION 语句中的 savepoint_name（即保存点的名称），savepoint_name 必须符合标识符规则，当条件回滚只影响事务的一部分时，可使用 savepoint_name。
- ☑ @savepoint_variable：是用户定义的、包含有效保存点名称的变量的名称，它必须用 char、varchar、nchar 或 nvarchar 数据类型声明变量。

在 ROLLBACK TRANSACTION 语句中用到了保存点，通常使用 SAVE TRANSACTION 语句在事

务内设置保存点。语法格式如下：

```
SAVE {TRAN | TRANSACTION} {savepoint_name | @savepoint_variable}[;]
```

参数说明如下。

- ☑ savepoint_name：是保存点的名称，它必须符合标识符规则。当条件回滚只影响事务的一部分时，可使用 savepoint_name。
- ☑ @savepoint_variable：是用户定义的、包含有效保存点名称的变量的名称，它必须用 char、varchar、nchar 或 nvarchar 数据类型声明变量。

16.3.4　事务的工作机制

下面将通过一个示例讲解事务的工作机制。

【例 16.03】 使用事务修改 Employee 表中的数据，并将指定的员工记录删除，SQL 语句及运行结果如图 16.3 所示。（**实例位置：资源包\源码\16\16.03**）

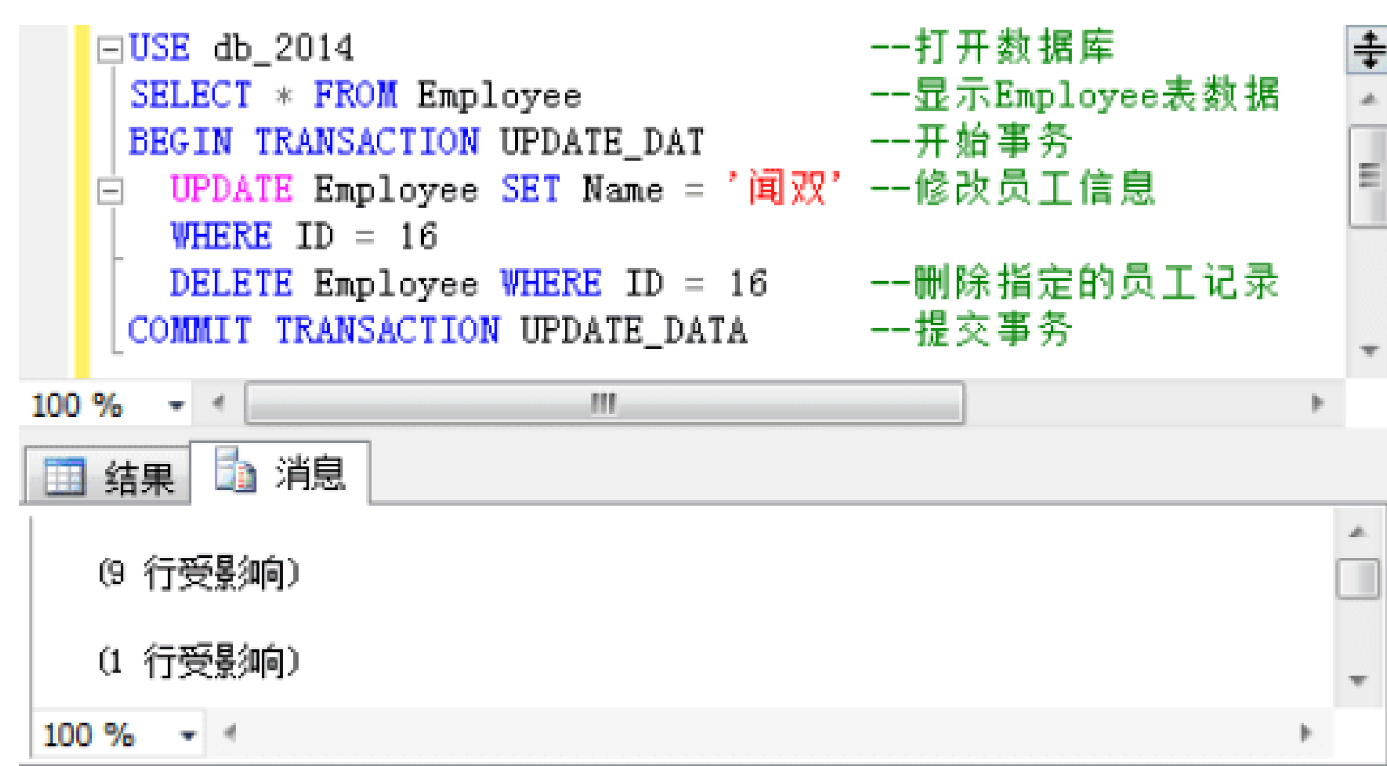


图 16.3　修改 Employee 表中的数据

SQL 语句如下：

```
USE db_2014                                        --打开数据库
SELECT * FROM Employee                             --显示 Employee 表数据
BEGIN TRANSACTION UPDATE_DAT                       --开始事务
  UPDATE Employee SET Name = '闻双'                --修改员工信息
  WHERE ID = 16
  DELETE Employee WHERE ID = 16                    --删除指定的员工记录
COMMIT TRANSACTION UPDATE_DATA                     --提交事务
```

例 16.03 中的事务的工作机制可以分为以下几点。

（1）当在代码中出现 BEGIN TRANSACTION 语句时，SQL Server 将会显示事务，并会给新事务分配一个事务 ID。

（2）当事务开始后，SQL Server 将会运行事务体语句，并将事务体语句记录到事务日志中。

（3）在内存中执行事务日志中所记录的事务体语句。

（4）当执行到 COMMIT 语句时会结束事务，同时事务日志也会被写到数据库的日志设备上，从而保证日志可以被恢复。

16.3.5　自动提交事务

自动提交事务是 SQL Server 默认的事务处理方式，当任何一条有效的 SQL 语句被执行后，它对数据库所做的修改都将会被自动提交，如果发生错误，则将会自动回滚并返回错误信息。

【例 16.04】 使用 INSERT 语句向数据库中添加 3 条记录，但由于添加了重复的主键，导致最后一条 INSERT 语句在编译时产生错误，从而使这条语句没有被执行，SQL 语句及运行结果如图 16.4 所示。**（实例位置：资源包\源码\16\16.04）**

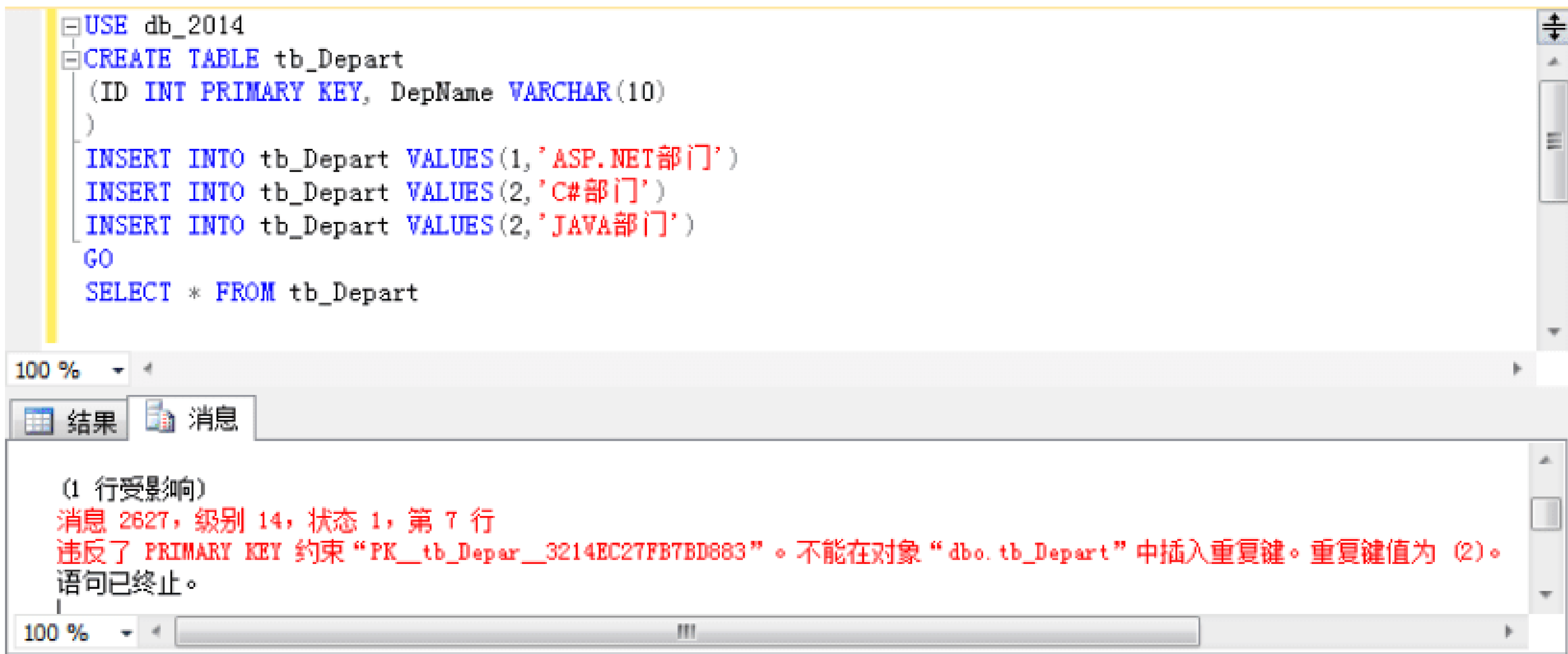


图 16.4　自动提交事务出现错误

SQL 语句如下：

```
USE db_2014                                         --打开数据库
CREATE TABLE tb_Depart                              --创建数据表
(ID INT PRIMARY KEY, DepName VARCHAR(10)
)
INSERT INTO tb_Depart VALUES(1,'ASP.NET 部门')       --插入记录
INSERT INTO tb_Depart VALUES(2,'C#部门')             --插入记录
INSERT INTO tb_Depart VALUES(2,'JAVA 部门')          --插入记录
GO
SELECT * FROM tb_Depart                             --检索记录
```

本示例中，SQL Server 将前两条记录添加到了指定的数据表中，而将第 3 条记录回滚，这是因为第 3 条记录出现编译错误并且不符合条件（主键不允许重复），所以被事务回滚。

16.3.6　事务的并发问题

事务的并发问题主要体现在丢失或覆盖更新、未确认的相关性（脏读）、不一致的分析（不可重复读）和幻象读 4 个方面，这些是影响事务完整性的主要因素。如果没有锁定且多个用户同时访问一个数据库，则当他们的事务同时使用相同的数据时可能会发生以上几种问题。下面分别进行说明。

（1）丢失更新

当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，会发生丢失更新问题。每个

事务都不知道其他事务的存在。最后的更新将重写由其他事务所做的更新，这样就会导致数据丢失。

例如，最初有一份原始的电子文档，文档人员 A 和 B 同时修改此文档，当修改完成之后保存时，最后修改完成的文档必将替换第一个修改完成的文档，那么就造成了数据丢失更新的后果。如果文档人员 A 修改并保存之后，文档人员 B 再进行修改则可以避免该问题。

（2）未确认的相关性（脏读）

如果一个事务读取了另外一个事务尚未提交的更新，则称为脏读。

例如，文档人员 B 复制了文档人员 A 正在修改的文档，并将文档人员 A 的文档发布，此后，文档人员 A 认为文档中存在着一些问题需要重新修改，此时文档人员 B 所发布的文档就将与重新修改的文档内容不一致。如果文档人员 A 将文档修改完成并确认无误的情况下，文档人员 B 再复制则可以避免该问题。

（3）不一致的分析（不可重复读）

当事务多次访问同一行数据，并且每次读取的数据不同时，将会发生不一致分析问题。不一致的分析与未确认的相关性类似，因为其他事务也正在更改该数据。然而，在不一致的分析中，事务所读取的数据是由进行了更改的事务提交的。而且，不一致的分析涉及多次读取同一行，并且每次信息都由其他事务更改，因而该行不可被重复读取。

例如，文档人员 B 两次读取文档人员 A 的文档，但在文档人员 B 读取时，文档人员 A 又重新修改了该文档中的内容，在文档人员 B 第二次读取文档人员 A 的文档时，文档中的内容已被修改，此时则发生了不可重复读的情况。如果文档人员 B 在文档人员 A 全部修改后读取文档，则可以避免该问题。

（4）幻象读

幻象读和不一致的分析有些相似，当一个事务的更新结果影响到另一个事务时，将会发生幻象读问题。事务第一次读的行范围显示出其中一行已不复存在于第二次读或后续读中，因为该行已被其他事务删除。同样，由于其他事务的插入操作，事务的第二次或后续读显示有一行已不存在于原始读中。

例如，文档人员 B 更改了文档人员 A 所提交的文档，但当文档人员 B 将更改后的文档合并到主副本时，却发现文档人员 A 已将新数据添加到该文档中。如果文档人员 B 在修改文档之前，没有任何人将新数据添加到该文档中，则可以避免该问题。

16.3.7　事务的隔离级别

当事务接受不一致的数据级别时被称为事务的隔离级别。如果事务的隔离级别比较低，会增加事务的并发问题，有效地设置事务的隔离级别可以降低并发问题的发生。

设置隔离数据可以使一个进程使用，同时还可以防止其他进程的干扰。设置隔离级别定义了 SQL Server 会话中所有 SELECT 语句的默认锁定行为，当锁定用作并发控制机制时，它可以解决并发问题。这使所有事务得以在彼此完全隔离的环境中运行，但是任何时候都可以有多个正在运行的事务。

在 SQL Server 中，可以使用 SET TRANSACTION ISOLATION LEVEL 语句来设置事务的隔离级别。

SET TRANSACTION ISOLATION LEVEL：控制由连接发出的所有 SELECT 语句的默认事务锁定行为。语法格式如下：

```
SET TRANSACTION ISOLATION LEVEL{READ COMMITTED | READ UNCOMMITTED | REPEATABLE
READ | SERIALIZABLE}
```

参数说明如下。

☑ READ COMMITTED：指定在读取数据时控制共享锁以避免脏读，但数据可在事务结束前更改，从而产生不可重复读取或幻象读取数据，该选项是 SQL Server 的默认值。

☑ READ UNCOMMITTED：执行脏读或 0 级隔离锁定，这表示不发出共享锁，也不接受排它锁，该选项的作用与在事务内所有语句中的所有表上设置 NOLOCK 相同，这是 4 个隔离级别中限制最小的级别。

☑ REPEATABLE READ：锁定查询中使用的所有数据以防止其他用户更新数据，但是其他用户可以将新的幻象读插入数据集，且幻象读包括在当前事务的后续读取中，因为并发低于默认隔离级别，所以应只在必要时才使用该选项。

☑ SERIALIZABLE：表示在数据集上放置一个范围锁，以防止其他用户在事务完成之前更新数据集或将行插入数据集内。

SQL Server 提供了 4 种事务的隔离级别，如表 16.1 所示。

表 16.1　事务的隔离级别

隔 离 级 别	脏　读	不可重复读	幻　象　读
READ UNCOMMITTED（未提交读）	是	是	是
READ COMMITTED（提交读）	否	是	是
REPEATABLE READ（可重复读）	否	否	是
SERIALIZABLE（可串行读）	否	否	否

SQL Server 的默认隔离级别为 READ COMMITTED，可以使用锁来实现隔离性级别。

（1）READ UNCOMMITTED（未提交读）

此隔离级别为隔离级别中最低的级别，如果将 SQL Server 的隔离级别设置为 READ UNCOMMITTED，则可以对数据执行未提交读或脏读，并且等同于将锁设置为 NOLOCK。

【例 16.05】　设置未提交读隔离级别。（**实例位置：资源包\源码\16\16.05**）

SQL 语句如下：

```
BEGIN TRANSACTION
UPDATE Employee SET Name = '章子婷'
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED        --设置未提交读隔离级别
COMMIT TRANSACTION
SELECT * FROM Employee
```

运行结果如图 16.5 所示。

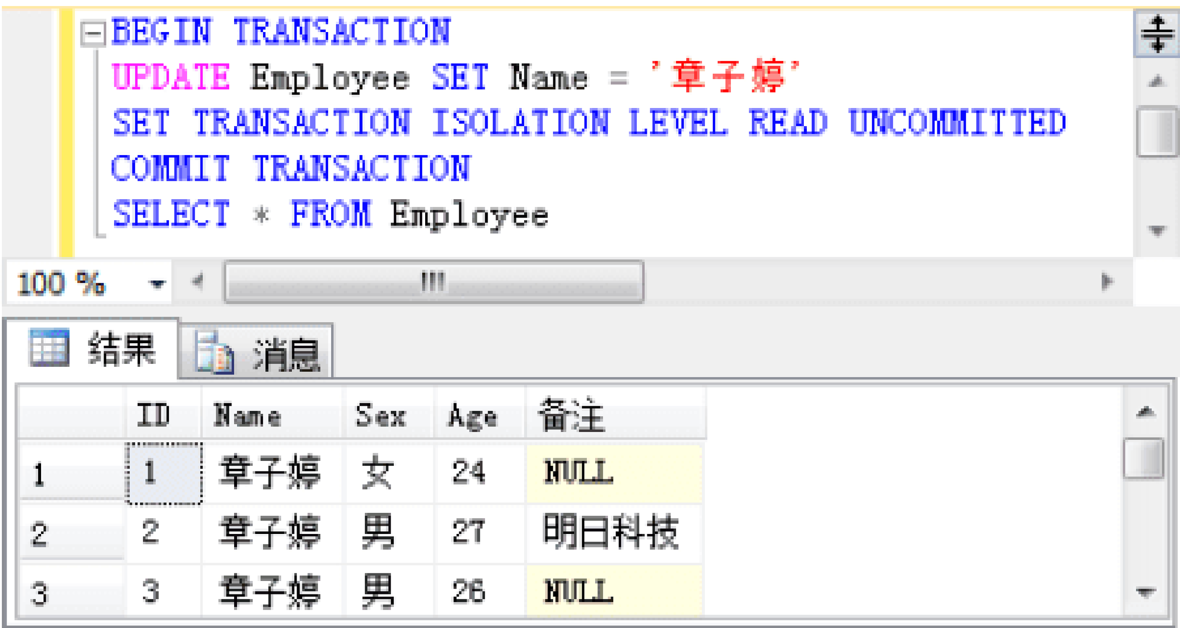


图 16.5　设置未提交读隔离级别

（2）READ COMMITTED（提交读）

此项隔离级别为 SQL 中默认的隔离级别，将事务设置为此级别，可以在读取数据时控制共享锁以避免脏读，从而产生不可重复读取或幻象读取数据。

【例 16.06】　设置提交读隔离级别。（**实例位置：资源包\源码\16\16.06**）

SQL 语句如下：

```
SET TRANSACTION ISOLATION LEVEL Read Committed
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
SET TRANSACTION ISOLATION LEVEL Read Committed              --设置提交读隔离级别
UPDATE Employee SET Name = '高丽'
```

运行结果如图 16.6 所示。

```
SET TRANSACTION ISOLATION LEVEL Read Committed
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
SET TRANSACTION ISOLATION LEVEL Read Committed
UPDATE Employee SET Name = '高丽'
```

结果　消息

	ID	Name	Sex	Age	备注
1	1	章子婷	女	24	NULL
2	2	章子婷	男	27	明日科技
3	3	章子婷	男	26	NULL

图 16.6　设置提交读隔离级别

（3）REPEATABLE READ（可重复读）

此项隔离级别增加了事务的隔离级别，将事务设置为此级别可以防止脏读、不可重复读和幻象读。

【例 16.07】　设置可重复读隔离级别。（**实例位置：资源包\源码\16\16.07**）

SQL 语句如下：

```
SET TRANSACTION ISOLATION LEVEL Repeatable Read
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
SET TRANSACTION   ISOLATION LEVEL Repeatable Read           --设置可重复读隔离级别
INSERT INTO Employee values ('18','张雨','男','22','明日科技')
```

运行结果如图 16.7 所示。

```
SET TRANSACTION ISOLATION LEVEL Repeatable Read
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
SET TRANSACTION  ISOLATION LEVEL Repeatable Read
INSERT INTO Employee values (18,'张雨','男','22','明日科技')
```

结果　消息

	ID	Name	Sex	Age	备注
1	1	高丽	女	24	NULL
2	2	高丽	男	27	明日科技
3	3	高丽	男	26	NULL

图 16.7　设置可重复读隔离级别

（4）SERIALIZABLE（可串行读）

此项隔离级别是所有隔离级别中限制最大的级别，它防止了所有的事务并发问题，此级别可以适用于绝对的事务完整性的要求。

【例 16.08】 设置可串行读隔离级别。（**实例位置：资源包\源码\16\16.08**）

SQL 语句如下：

```
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
SET TRANSACTION ISOLATION LEVEL Serializable      --设置可串行读
DELETE FROM   Employee   WHERE ID = '1'
```

运行结果如图 16.8 所示。

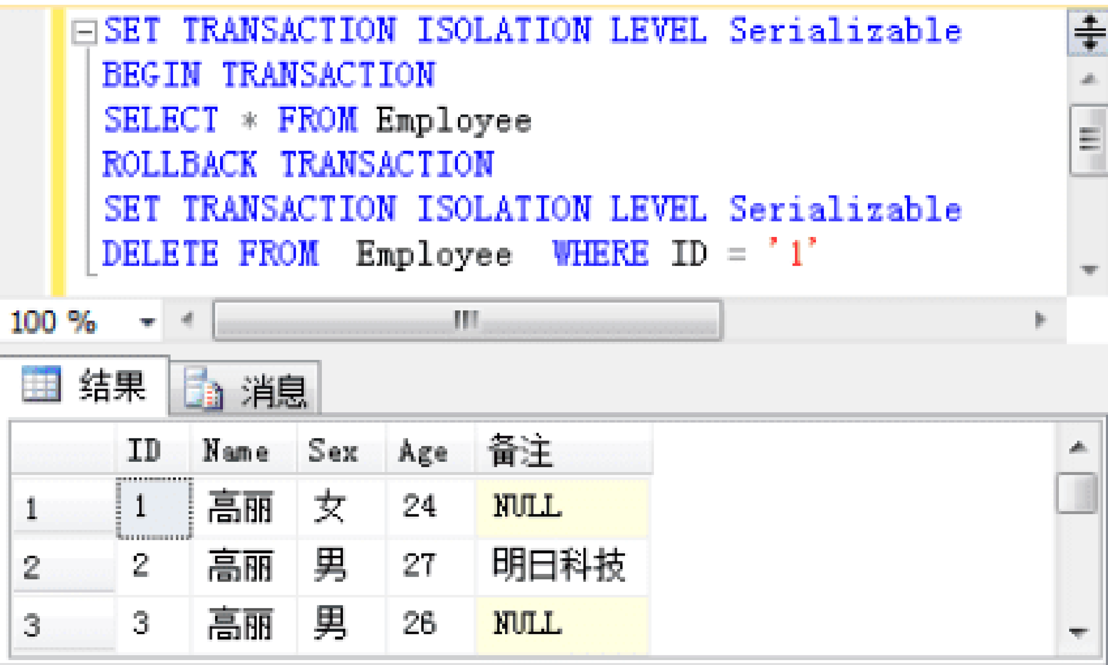


图 16.8 设置可串行读



16.4 锁

锁是一种机制，用于防止一个过程在对象上进行操作时，同某些已经在该对象上完成的事情发生冲突。锁可以防止事务的并发问题，如丢失更新、脏读（Dirty Read）、不可重复读（NO-Repeatable Read）和幻象（Phantom）等问题。本节主要介绍锁的机制、模式等。

16.4.1 SQL Server 锁机制

锁在数据库中是一个非常重要的概念，锁可以防止事务的并发问题，在多个事务访问下能够保证数据库完整性和一致性。例如，当多个用户同时修改或查询同一个数据库中的数据时，可能会导致数据不一致的情况，为了控制此类问题的发生，SQL Server 引入了锁机制。

在各类数据库中所使用的锁机制基本是一致的，但也有个别不同。当使用数据库时，SQL Server 采用系统来管理锁，例如，当用户向 SQL Server 发送某些命令时，SQL Server 将通过满足锁的条件为数据库加上适当的锁，这也就是动态加锁。

在用户对数据库没有特定要求的情况下，通过系统自动管理锁即可满足基本的使用要求，相反，如果用户在数据库的完整性和一致性方面有特殊的要求，则需要使用锁来实现用户的要求。

16.4.2　锁模式

锁具有模式属性，它用于确定锁的用途，如表 16.2 所示。

表 16.2　锁模式

锁　模　式	描　　述
共享（S）	用于不更改或不更新数据的操作（只读操作），如 SELECT 语句
更新（U）	用于可更新的资源中。防止当多个会话在读取、锁定以及随后可能进行的资源更新时发生常见形式的死锁
排它（X）	用于数据修改操作，如 INSERT、UPDATE 或 DELETE。确保不会同时出现同一资源进行多重更新
意向	用于建立锁的层次结构。意向锁的类型为意向共享（IS）、意向排它（IX）以及与意向排它共享（SIX）
架构	在执行依赖于表架构的操作时使用。架构锁的类型为架构修改（Sch-M）和架构稳定性（Sch-S）
大容量更新（BU）	向表中大容量复制数据并指定了 TABLOCK 提示时使用

（1）共享锁

共享锁用于保护读取的操作，它允许多个并发事务读取其锁定的资源。在默认情况下，数据被读取后，SQL Server 立即释放共享锁并可以对释放的数据进行修改。例如，执行查询 SELECT * FROM table1 时，首先锁定第一页，直到读取后的第一页被释放锁时才锁定下一页。但是，事务隔离级别连接的选项设置和 SELECT 语句中的锁定设置都可以改变 SQL Server 的这种默认设置。例如，SELECT * FROM table1 HOLDLOCK 在表的查询过程中一直保存锁定，直到查询完成才释放锁定。

（2）更新锁

更新锁在修改操作的初始化阶段用来锁定要被修改的资源。它避免使用共享锁造成的死机现象，因为使用共享锁修改数据时，如果同时有两个或多个事务同时对一个事务申请了共享锁，而这些事务都将共享锁升级为排它锁，这时，这些事务都不会释放共享锁而是一直等待对方释放，这样很容易造成死锁。如果一个数据在修改前直接申请更新锁并在修改数据时升级为排它锁，就可以避免死机现象。

（3）排它锁

排它锁是为修改数据而保留的，它锁定的资源既不能读取也不能修改。

（4）意向锁

意向锁表示 SQL Server 在资源的底层获得共享锁或排它锁的意向。例如，表级的共享意向锁表示事务意图将排它锁释放到表的页或行中。意向锁又可以分为共享意向锁、独占意向锁和共享式独占意向锁。共享意向锁表明事务意图锁定底层资源上放置共享锁来读取数据。独占意向锁表明事务意图锁定底层资源上放置排它锁来修改数据。共享式独占意向锁表明事务允许其他事务使用共享锁来读取顶层资源，并意图在该资源底层上放置排它锁。

（5）架构锁

架构锁用于执行依赖于表架构的操作。构架锁又分为架构修改（Sch-M）锁和架构稳定性 （Sch-S）锁。架构修改（Sch-M）锁表示执行表的数据定义语言（DDL）操作；架构稳定性（Sch-S）锁表示不阻塞任何事务锁并包括排它锁。在编译查询时，其他事务（包括在表上有排它锁的事务）都能继续运

行，但不能在表上执行 DDL 操作。

（6）大容量更新锁

向表中大容量复制数据并且指定 tablock 提示，或者在 sp_tableoption 设置 table lock on bulk 表选项时而使用大容量更新锁。大容量更新锁允许进程将数据并发地大容量复制到同一表中，同时防止其他不进行大容量复制数据的进程访问该表。

16.4.3 锁的粒度

为了优化数据的并发性，可以使用 SQL Server 中锁的粒度，它可以锁定不同类型的资源。为了使锁定的成本减至最低，SQL Server 自动将资源锁定在适合任务的级别。如果锁的粒度大，则并发性高且开销大，如果锁的粒度小，则并发性低且开销小。

SQL Server 支持的锁粒度如表 16.3 所示。

表 16.3 锁的粒度

锁 大 小	描 述
行锁（RID）	行标识符。用于单独锁定表中的一行，这是最小的锁
键锁	锁定索引中的节点。用于保护可串行事务中的键范围
页锁	锁定 8KB 的数据页或索引页
扩展盘区锁	锁定相邻的 8 个数据页或索引页
表锁	锁定整个表
数据库锁	锁定整个数据库

（1）行锁（RID）

行锁为锁的粒度当中最小的资源。行锁就是指事务在操作数据的过程中，锁定一行或多行的数据，其他事务不能同时处理这些行的数据。行锁占用的数据资源最小，所以在事务的处理过程中，允许其他事务操作同一个表中的其他数据。

（2）页锁

页锁是指事务在操作数据的过程中，一次锁定一页。在 SQL Server 中 25 个行锁可以升级为一个页锁，当此页被锁定后，其他事务就不能够操作此页数据，即使只锁定一条数据，那么其他事务也不能够对此页数据进行操作。从而可以看出页锁与其行锁相比，页锁占用的数据资源要多。

（3）表锁

表锁是指事务在操作数据的过程中，锁定了整个数据表。当整个数据表被锁定后，其他事务不能够使用此表中的其他数据。表锁的特点是使用事务处理的数据量大，并且使用较少的系统资源。但是当使用表锁时，如果所占用的数据量大，那么将会延迟其他事务的等待时间，从而降低了系统的并发性能。

（4）数据库锁

数据库锁可锁定整个数据库，可防止任何事务或用户对此数据库进行访问，数据库锁是一种比较特殊的锁，它可以控制整个数据库的操作。

数据库锁可用于在进行数据恢复操作，当进行此操作时，就可以防止其他用户对此数据库进行各种操作。

16.4.4　查看锁

在 SQL Server 2014 中，查看锁的相关信息，通常使用 sys.dm_tran_locks 动态管理视图，下面来看一个示例。

【例 16.09】　使用 sys.dm_tran_locks 动态管理视图查看活动锁的信息，SQL 语句及运行结果如图 16.9 所示。（**实例位置：资源包\源码\16\16.09**）

图 16.9　显示锁信息

SQL 语句如下：

```
select * from sys.dm_tran_locks
```

另外，在早期的版本中，通常使用 sp_lock 储存过程来查看，在 SQL Server 2014 数据库中，该存储过程同样适用。

语法格式如下：

```
sp_lock [[@spid1 =] 'spid1'] [,[@spid2 =] 'spid2']
```

参数说明如下。

☑　[@spid1 =] 'spid1'：表示来自 master.dbo.sysprocesses 的 SQL Server 进程 ID 号，spid1 的数据类型为 int，默认值为 NULL，执行 sp_who 可获取有关该锁的进程信息，如果没有指定 spid1，则显示所有锁的信息。

☑　[@spid2 =] 'spid2'：用于检查锁信息的另一个 SQL Server 进程 ID 号，spid2 的数据类型为 int，默认设置为 NULL，spid2 为可以与 spid1 同时拥有锁的另一个 spid，用户可以获取有关它的信息。

16.4.5　死锁

当两个或多个线程之间有循环相关性时，将会产生死锁。死锁是一种可能发生在任何多线程系统中的状态，而不仅仅发生在关系数据库管理系统中。多线程系统中的一个线程可能获取一个或多个资源（如锁）。如果正获取的资源当前为另一线程所拥有，则第一个线程可能必须等待拥有线程释放目标资源，这时就说等待线程在哪个特定资源上与拥有线程有相关性。

在数据库系统中，如果多个进程分别锁定了一个资源，并又要访问已经被锁定的资源，则此时就会产生死锁，同时也会导致多个进程都处于等待的状态。在事务提交或回滚之前两个线程都不能释放

资源，而且它们因为正等待对方拥有的资源而不能提交或回滚事务。

例如，事务 A 的线程 T1 具有 Supplier 表上的排它锁。事务 B 的线程 T2 具有 Part 表上的排它锁，并且之后需要 Supplier 表上的锁。事务 B 无法获得这一锁，因为事务 A 已拥有它。事务 B 被阻塞，等待事务 A。然而，事务 A 需要 Part 表的锁，但又无法获得锁，因为事务 B 将它锁定了。

程序示意图如图 16.10 所示。

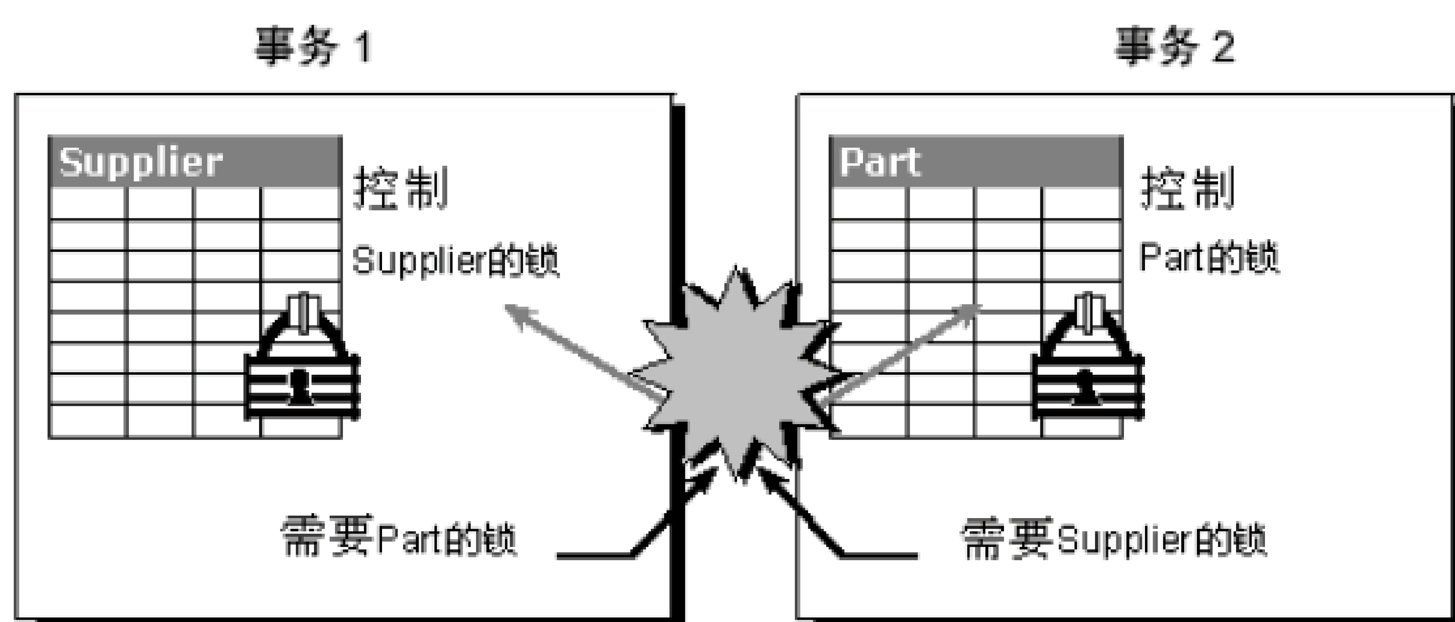


图 16.10　死锁示意图

在图 16.10 中，对于 Part 表锁资源，线程 T1 在线程 T2 上具有相关性。同样，对于 Supplier 表锁资源，线程 T2 在线程 T1 上具有相关性。因为这些相关性形成了一个循环，所以在线程 T1 和线程 T2 之间存在死锁。


说明

事务在提交或回滚之前不能释放持有的锁。因为事务需要对方控制的锁才能继续操作，所以它们不能提交或回滚。BEGIN TRANSACTION 与 COMMIT TRANSACTION 之间的语句，可以是任何对数据库进行修改的语句。

可以使用 LOCK_timeout 来设置程序请求锁定的最长等待时间，如果一个锁定请求等待超过了最长等待时间，那么该语句将被自动取消。LOCK_timeout 语句主要用于自定义锁超时。语法格式如下：

```
SET Lock_timeout[timeout_period]
```

参数 timeout_period 以毫秒为单位，值为-1（默认值）时表示没有超时期限（即无限期等待）。当锁等待超过超时值时，将返回错误。值为 0 时表示根本不等待，并且一遇到锁就返回信息。

【例 16.10】 将锁超时期限设置为 5000 毫秒。（**实例位置：资源包\源码\16\16.10**）

SQL 语句如下：

```
SET Lock_timeout 5000
```


视频讲解

16.5　分布式事务处理

在前面的学习中我们已经了解，事务是单个的工作单元，而分布式事务则是跨越两个或多个数据库的。本节主要介绍分布式事务、如何创建分布式事务与分布式事务处理协调器。

16.5.1　分布式事务简介

在事务处理中，涉及一个以上数据库的事务被称为分布式事务。分布式事务跨越两个或多个称为资源管理器的服务器。如果分布式事务由 Microsoft 分布式事务处理协调器（MS DTC）这类事务管理器或其他支持 X/Open XA 分布式事务处理规范的事务管理器进行协调，则 SQL Server 可以作为资源管理器运行。

16.5.2　创建分布式事务

保证数据的完整性十分重要，要保证数据的完整性，就要在事务处理中保证事务的原子性，在分布式事务处理中主要使用了分布式事务处理协调器，一台服务器上只能运行一个处理协调器实例，必须启动了分布式事务处理协调器才能执行分布式事务。否则事务就会失败。

下面通过一个示例讲解如何创建一个分布式事务。

【例 16.11】　利用分布式事务对链接的远程数据源 MR 的 db_CSharp 数据库中的 Employee 表和本地 Employee 表进行修改。（**实例位置：资源包\源码\16\16.11**）

SQL 语句如下：

```
SET Xact_Abort ON
BEGIN DISTRIBUTED TRANSACTION
UPDATE Employee SET Name = '星星' WHERE ID = 1
UPDATE [MR].[db_CSharp].[dbo].[Employee] SET Name = '婷子' WHERE ID = 1
COMMIT TRANSACTION
```



注意

本示例在执行分布式事务时，须启动服务项 Distributed Transaction Coordinator。

在上段代码中使用了 Xact_Abort 语句，此语句可实现当出现错误时回滚当前 Transact-SQL 命令，在 Xact_Abort 语句执行之后，任何运行时语句错误都将导致当前事务自动回滚。编译错误（如语法错误）不受 Xact_Abort 语句的影响。



说明

分布式事务处理要保证事务的原子性，即在事务执行过程中发生错误时，已更新操作必须可以回滚，否则事务数据库就会处于不一致状态。

16.5.3　分布式事务处理协调器

分布式事务处理协调器（DTC）系统服务负责协调跨计算机系统和资源管理器分布的事务，如数

据库、消息队列、文件系统和其他事务保护资源管理器。如果事务性组件是通过 COM+配置的，就需要 DTC 系统服务。消息队列（也称作 MSMQ）中的事务性队列和 SQL Server 跨多系统运行也需要 DTC 系统服务。

16.6 小　　结

本章主要对 SQL Server 2014 中的事务进行详细讲解，具体讲解过程中，首先介绍事务的概念，让读者对什么是事务有一个清晰的了解；然后讲解了显式与隐式事务、如何使用事务、事务的工作机制及并发事务的使用等高级内容；最后还讲解了与事务关系密切的锁和分布式事务处理等内容。通过本章的学习，读者应该熟练掌握事务的使用，并能够使用事务解决数据库开发中遇到的问题。

第17章

SQL Server 高级开发

（视频讲解：14 分钟）

本章主要介绍 SQL Server 2014 的高级应用，包括用户自定义函数和实现交叉表查询。通过本章的学习，读者可以创建和管理用户自定义函数，可以使用 PIVOT、UNPIVOT 以及 CASE 实现交叉表查询。

学习摘要：

- **创建用户自定义函数**
- **修改、删除用户自定义函数**
- **使用 PIVOT 和 UNPIVOT 实现交叉表查询**
- **使用 CASE 实现交叉表查询**


视频讲解

17.1 用户自定义函数

SQL Server 2014 还可以根据用户需要来自定义函数，以便用在允许使用系统函数的任何地方。

用户自定义函数有两种方法：一种是利用 SQL Server Management Studio 管理工具直接创建；另一种是利用代码创建。

17.1.1 创建用户自定义函数

用 SQL Server Management Studio 管理工具直接创建用户自定义函数的具体步骤如下。

（1）选择“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 命令，打开 SQL Server Management Studio 管理工具窗口。

展开服务器组，选择要在其中创建用户自定义数据类型的数据库。展开“可编程性”→“函数”节点，单击鼠标右键，在弹出的快捷菜单中选择“新建”命令，如图 17.1 所示。

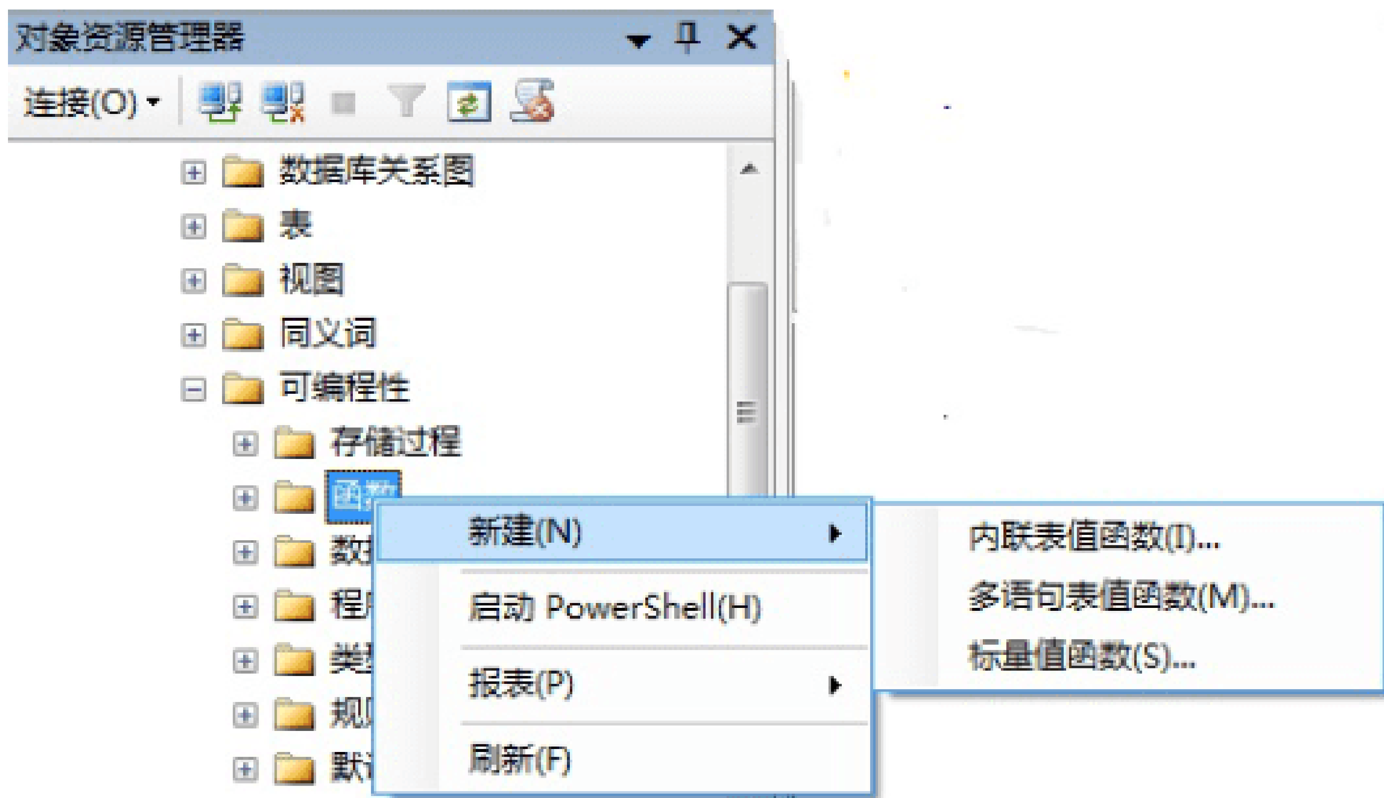


图 17.1　创建自定义函数

（2）根据函数的返回值不同，函数分为内联表值函数、多语句表值函数和标量值函数，用户可以根据需要任选其一。

（3）选择其中一种自定义函数后，打开一个创建自定义函数的数据库引擎查询模板，只需要修改其相应的参数即可。

17.1.2 使用 Transact-SQL 语言创建用户自定义函数

（1）创建自定义函数

利用 Transact-SQL 创建函数的语法格式如下：

```
CREATE FUNCTION 函数名 (@parameter 变量类型 [,@parameter 变量类型])
RETURNS 参数 AS
BEGIN
 命令行或程序块
END
```

函数可以有 0 个或若干个输入参数，但必须有返回值，RETURNS 后面就是设置函数的返回值类型。

用户自定义函数为标量值函数或表值函数。如果 RETURNS 子句指定了一种标量数据类型，则函数为标量值函数；如果 RETURNS 子句指定 TABLE，则函数为表值函数。根据函数主体的定义方式，表值函数可分为内联函数和多语句函数。

例如，创建一个自定义标量值函数 max1，max1 函数的功能是返回两个数中的最大值。SQL 语句如下：

```
CREATE FUNCTION max1( @x int , @y int)
RETURNS int AS
BEGIN
IF @x<@y
SET @x=@y
RETURN @x
END
```

（2）调用自定义函数

Transact-SQL 调用函数的语法格式如下：

```
PRINT dbo.函数（[实参])
```

或：

```
SELECT dbo.函数（[实参])
```

dbo 是系统自带的一个公共用户名。

例如，调用上个例子创建的 max1 函数，输出@a 和@b 两个变量中的最大值。SQL 语句如下：

```
DECLARE @a int , @b int
SET @a=10
SET @b=20
PRINT dbo.max1(@a , @b)
```

运行结果是：20。

【例 17.01】　创建 find 表的自定义函数。（**实例位置：资源包\源码\17\17.01**）

创建一个名称是 find 的内联表值函数，其功能是在 tb_basicMessage 表中，根据输入的 age 进行查询。SQL 语句如下：

```
CREATE FUNCTION find(@x int)
RETURNS TABLE
AS
RETURN(SELECT * FROM tb_basicMessage WHERE age>@x)
```

在 tb_basicMessage 表中，查询 age 大于所输入的参数的员工信息。SQL 语句如下：

```
USE db_supermarket
SELECT * FROM find (27)
```

查询结果如图 17.2 所示。

	id	name	age	sex	dept	headship
1	8	小葛	29	男	1	1
2	16	张三	30	男	1	5
3	23	小开	30	男	4	4

图 17.2　用 find 函数查询的结果

17.1.3　修改、删除用户自定义函数

1．修改自定义函数

利用 Transact-SQL 修改函数的语法格式如下：

```
ALTER FUNCTION 函数名 (@parameter 变量类型 [,@parameter 变量类型])
RETURNS 参数 AS
BEGIN
 命令行或程序块
END
```

修改函数与创建函数几乎相同，将 create 改成 alter 即可。

2．删除自定义函数

删除自定义函数的 Transact-SQL 语法格式如下：

```
DROP FUNCTION 函数名
```

例如，删除 tb_basicMessage 表的自定义函数：

```
DROP FUNCTION FIND
```



17.2　使用 SQL Server 2014 实现交叉表查询

17.2.1　使用 PIVOT 和 UNPIVOT 实现交叉表查询

PIVOT 和 UNPIVOT 运算符是 SQL Server 2014 新增的功能。通过 PIVOT 和 UNPIVOT 就完全可以实现交叉表的查询，用 PIVOT 和 UNPIVOT 编写更简单，更易于理解。

在查询的 FROM 子句中使用 PIVOT 和 UNPIVOT，可以对一个输入表值表达式执行某种操作，以获得另一种形式的表。PIVOT 运算符将输入表的行旋转为列，并能同时对行执行聚合运算。而 UNPIVOT 运算符则执行与 PIVOT 运算符相反的操作，它将输入表的列旋转为行。PIVOT 和 UNPIVOT 的语法格式如下：

```
[FROM {<table_source>} [,...n]]
<table_source> ::= {
```

```
  table_or_view_name [[AS] table_alias]
  <pivoted_table> | <unpivoted_table>}
<pivoted_table> ::=table_source PIVOT <pivot_clause> table_alias
<pivot_clause> ::=(aggregate_function (value_column)
  FOR pivot_column
    IN <column_list>)
<unpivoted_table> :: = table_source UNPIVOT <unpivot_clause> table_alias
<unpivot_clause> :: = value_column FOR pivot_column IN <column_list>
<column_list> :: = column_name [, ...] table_source PIVOT <pivot_clause>
```

参数说明如表 17.1 所示。

表 17.1　PIVOT 和 UNPIVOT 运算符的参数说明

参　数	描　述
<table_source>	指定要在 Transact-SQL 语句中使用的表、视图或派生表源（有无别名均可）。虽然语句中可用的表源个数的限值根据可用内存和查询中其他表达式的复杂性而有所不同，但一个语句中最多可使用 256 个表源。单个查询可能不支持最多有 256 个表源。在该参数中可将 table 变量指定为表源。表源在 FROM 关键字后的顺序不影响返回的结果集。如果 FROM 子句中出现重复的名称，SQL Server 2014 会返回错误消息
table_or_view_name	表或视图的名称。如果表或视图位于正在运行 SQL Server 实例的同一计算机上的另一个数据库中，请按照 database.schema.object_name 形式使用完全限定名。如果表或视图不在链接服务器上的本地服务器中，请按照 linked_server.catalog.schema.object 形式使用 4 个部分的名称。如果由 4 部分组成的表或视图名称的服务器部分使用的是 OPENDATASOURCE 函数，则该名称也可用于指定表源。有关该函数的详细信息，请参阅 OPENDATASOURCE（Transact-SQL）
[AS] table_alias	table_source 的别名，别名可带来使用上的方便，也可用于区分自联结或子查询中的表或视图。别名往往是一个缩短了的表名，用于在联结中引用表的特定列。如果联结中的多个表中存在相同的列名，SQL Server 要求使用表名、视图名或别名来限定列名。如果定义了别名则不能使用表名。如果使用派生表、行集或表值函数或者运算符子句（如 PIVOT 或 UNPIVOT），则在子句结尾处必需的 table_alias 是所有返回列（包括组合列）的关联表名
table_source PIVOT <pivot_clause>	指定基于 table_source 对 pivot_column 进行透视。table_source 是表或表表达式。输出是包含 table_source 中 pivot_column 和 value_column 列之外的所有列的表。table_source 中 pivot_column 和 value_column 列之外的列被称为透视运算符的组合列。PIVOT 对输入表执行组合列的分组操作，并为每个组返回一行。此外，input_table 的 pivot_column 中显示的 column_list 中指定的每个值，输出中都对应一列
aggregate_function	系统或用户定义的聚合函数。聚合函数应该对空值固定不变。对空值固定不变的聚合函数在求聚合值时不考虑组中的空值。不允许使用 COUNT(*)系统聚合函数
value_column	PIVOT 运算符的值列。与 UNPIVOT 一起使用时，value_column 不能是输入 table_source 中的现有列的名称
FOR pivot_column	PIVOT 运算符的透视列。pivot_column 必须属于可隐式或显式转换为 nvarchar()的类型。此列不能为 image 或 rowversion。使用 UNPIVOT 时，pivot_column 是从 table_source 中提取的输出列的名称。table_source 中不能有该名称的现有列
IN <column_list>	在 PIVOT 子句中，列出 pivot_column 中将成为输出表的列名的值。该列表不能指定被透视的输入 table_source 中已存在的任何列名。在 UNPIVOT 子句中，列出 table_source 中将被提取到单个 pivot_column 中的列

续表

参　　数	描　　述
table_alias	输出表的别名。必须指定 pivot_table_alias
UNPIVOT <unpivot_clause>	指定输入表从 column_list 中的多个列缩减为名为 pivot_column 的单个列

例如，如图 17.3 所示的商品表就是一个典型的交叉表，其中“数量”和“月份”可以继续添加。但是，这种格式在进行数据表存储的时候并不容易管理。例如，存储如图 17.4 所示的表格数据时，通常需要设计成如图 17.5 所示的结构。这样就带来一个问题，用户既希望数据容易管理，又希望能够生成一种方便阅读的表格数据。恰好 PIVOT 能够满足这两个条件。

数量 / 商品	一月	二月	三月	…
商品 1				
商品 2				
商品 3				
…				

图 17.3　商品表

商品名称	销售数量	月份

图 17.4　商品表结构

现设计如图 17.5 所示的 sp（商品）表，其中有商品名称、销售数量和月份列，并存储相应的数据。SQL 语句如下：

```
USE STUDENT
SELECT  商品名称,a.[9] AS [九月],a.[10] AS [十月],a.[11] AS [十一月],a.[12] AS [十二月]
FROM sp
PIVOT(SUM(销售数量) FOR 月份 IN([9],[10],[11],[12] )) AS a
```

其中，sp 是输入表，月份是透视列（pivot_column），销售数量是值列（value_column）。上面的语句将按下面的步骤获得输出结果集。

（1）PIVOT 首先按值列之外的列（商品名称和月份）对输入表 sp 进行分组汇总，类似执行下面的 SQL 语句：

```
USE STUDENT
SELECT  商品名称, 月份, SUM(销售数量) AS total
FROM sp
GROUP BY  商品名称, 月份
```

执行上述 SQL 语句将得到如图 17.6 所示的中间结果集。

	商品名称	销售数量	月份
1	李小葱	888	9
2	周木人专辑	777	9
3	国产E601	564	11
4	920演唱会DVD	333	10
5	李小葱专辑	28888	10
6	周木人专辑	778	10
7	国产E601	2478	12
8	920演唱会DVD	6666	11
9	920演唱会DVD	8888	11
10	李小葱专辑	9999	11

图 17.5　sp 表

	商品名称	月份	total
1	李小葱	9	888
2	周木人专辑	9	777
3	920演唱会DVD	10	333
4	李小葱专辑	10	28888
5	周木人专辑	10	778
6	920演唱会DVD	11	15554
7	国产E601	11	564
8	李小葱专辑	11	9999
9	国产E601	12	2478

图 17.6　sp 表经过分组汇总后的结果

（2）PIVOT 根据“FOR 月份 IN”指定的值 9，10，11，12 在结果集中建立名为 9，10，11，12 的列，然后在中间结果集从月份列中取出相符合的值，分别放置到 9，10，11，12 列。此时得到别名为 a（见语句中 AS a 的指定）的结果集，如图 17.7 所示。

（3）最后根据“SELECT 商品名称, a.[9] AS [九月],a.[10] AS [十月],a.[11] AS [十一月],a.[12] AS [十二月] FROM”的指定，从别名是 a 的结果集中检索数据，并分别将名为 9，10，11，12 的列在最终结果集中重新命名为：九月、十月、十一月、十二月。这里需要注意的是 FROM 的含义，其表示在通过 PIVOT 关系运算符得到的 a 结果集中检索数据，而不是从 sp 表中检索数据。最终得到的结果集如图 17.8 所示。

	商品名称	九月	十月	十一月	十二月
1	920演唱会DVD	NULL	333	15554	NULL
2	国产E601	NULL	NULL	564	2478
3	李小葱专辑	888	28888	9999	NULL
4	周木人专辑	777	778	NULL	NULL

图 17.7　使用“for 月份 in([9] , [10] , [11] , [12])”后得到的结果集

	商品名称	九月	十月	十一月	十二月
1	920演唱会DVD	NULL	333	15554	NULL
2	国产E601	NULL	NULL	564	2478
3	李小葱专辑	888	28888	9999	NULL
4	周木人专辑	777	778	NULL	NULL

图 17.8　由 sp 表经行转列得到的最终结果集

UNPIVOT 与 PIVOT 执行几乎完全相反的操作，将列转换为行。假设如图 17.8 所示的结果集存储在一个名为 temp 的表中，现在需要将列标识符“九月”“十月”“十一月”“十二月”转换到对应于相应商品名称的行值中。这意味着必须另外标识两个列，一个用于存储月份，一个用于存储销售数量。为了便于理解，仍旧将这两个列命名为月份和销售数量。SQL 语句如下：

```
USE STUDENT
SELECT * FROM temp
UNPIVOT(销售数量
FOR 月份 in([九月],[十月],[十一月],[十二月])) AS b
```

运行上述 SQL 语句后的结果集如图 17.9 所示。

	商品名称	销售数量	月份
1	920演唱会DVD	333	十月
2	920演唱会DVD	15554	十一月
3	国产E601	564	十一月
4	国产E601	2478	十二月
5	李小葱专辑	888	九月
6	李小葱专辑	28888	十月
7	李小葱专辑	9999	十一月
8	周木人专辑	777	九月
9	周木人专辑	778	十月

图 17.9　使用 UNPIVOT 得到的结果集

但是，UNPIVOT 并不完全是 PIVOT 的逆操作，由于在执行 PIVOT 过程中，数据已经被进行了分组汇总，所以使用 UNPIVOT 有时并不会重现原始表值表达式的结果。

1．用 PIVOT 举例

【例 17.02】　使用 PIVOT 运算符实现交叉表查询。（**实例位置：资源包\源码\17\17.02**）

在 sp 表中，按“商品名称”实现交叉表查询。结果表显示各商品在各月的销售情况。SQL 语句如下：

```
USE STUDENT
SELECT * FROM sp PIVOT(SUM(销售数量) FOR 商品名称 IN([李小葱专辑],[周木人专辑],[国产 E601],[920 演唱会 DVD] )) AS 统计
```

实现的结果如图 17.10 所示。

有时还需要根据表的其他字段进行交叉查询。例如，在 sp 表中，按“月份”交叉查询。逐月进行聚合计算。SQL 语句如下：

```
USE STUDENT
SELECT 商品名称,a.[9] AS [九月],a.[10] AS [十月],a.[11] AS [十一月],a.[12] AS [十二月] FROM sp
PIVOT(SUM(销售数量) FOR 月份 IN([9],[10],[11],[12] )) AS a
```

实现的结果如图 17.11 所示。

	月份	李小葱专辑	周木人专辑	国产E601	920演唱会DVD
1	9	888	777	NULL	NULL
2	10	28888	778	NULL	333
3	11	9999	NULL	564	15554
4	12	NULL	NULL	2478	NULL

图 17.10　sp 表按商品名称交叉查询

	商品名称	九月	十月	十一月	十二月
1	920演唱会DVD	NULL	333	15554	NULL
2	国产E601	NULL	NULL	564	2478
3	李小葱专辑	888	28888	9999	NULL
4	周木人专辑	777	778	NULL	NULL

图 17.11　sp 表按月份交叉查询

2. 用 UNPIVOT 举例

UNPIVOT 是 PIVOT 的逆操作。假设如图 17.12 所示的结果集存储在结果表 temp1 中，如图 17.13 所示的结果集存储在结果表 temp2 中。

【例 17.03】 使用 UNPIVOT 运算符实现交叉表查询。（**实例位置：资源包\源码\17\17.03**）

用 UNPIVOT 实现把 temp1 表中的列标识李小葱专辑、周木人专辑、国产 E601 和 920 演唱会 DVD 转换到商品名称的行值中。相当于示例 PIVOT 的逆操作。SQL 语句如下：

```
USE STUDENT
SELECT * FROM temp1 UNPIVOT(销售数量 FOR 商品名称 IN([李小葱专辑],[周木人专辑],[国产 E601],[920 演唱会 DVD] )) AS a
```

实现的结果如图 17.12 所示。

用 UNPIVOT 实现把 temp2 中的列标识 9 月份、10 月份、11 月份和 12 月份列标识名称的行值中。相当于把示例的 PIVOT 实现逆操作。SQL 语句如下：

```
USE STUDENT
SELECT * FROM temp2 UNPIVOT(销售数量 FOR 月份 IN([九月],[十月],[十一月],[十二月] )) AS a
```

实现的结果如图 17.13 所示。

	月份	销售数量	商品名称
1	9	888	李小葱专辑
2	9	777	周木人专辑
3	10	28888	李小葱专辑
4	10	778	周木人专辑
5	10	333	920演唱会DVD
6	11	9999	李小葱专辑
7	11	564	国产E601
8	11	15554	920演唱会DVD
9	12	2478	国产E601

图 17.12　UNPIVOT 对 temp1 表实现逆操作

	商品名称	销售数量	月份
1	920演唱会DVD	333	十月
2	920演唱会DVD	15554	十一月
3	国产E601	564	十一月
4	国产E601	2478	十二月
5	李小葱专辑	888	九月
6	李小葱专辑	28888	十月
7	李小葱专辑	9999	十一月
8	周木人专辑	777	九月
9	周木人专辑	778	十月

图 17.13　UNPIVOT 对 temp2 实现逆操作

17.2.2　使用 CASE 实现交叉表查询

利用 CASE 语句可以返回多个可能结果的表达式。CASE 具有简单 CASE 和 CASE 查询两种函数

格式。下面介绍简单 CASE 语句的语法。

简单 CASE 语句：将某个表达式与一组简单表达式进行比较以确定结果。其语法格式如下：

```
CASE input_expression
    WHEN when_expression THEN result_expression
        [...n]
    [
        ELSE else_result_expression
    END
```

参数说明如下。

- ☑ input_expression：是使用简单 CASE 格式时所计算的表达式。input_expression 是任何有效的 SQL Server 表达式。
- ☑ WHEN when_expression：使用简单 CASE 格式时 input_expression 所比较的简单表达式。when_expression 是任意有效的 SQL Server 表达式。input_expression 和每个 when_expression 的数据类型必须相同，或者是隐性转换。
- ☑ n：占位符，表明可以使用多个 WHEN when_expression THEN result_expression 子句或 WHEN Boolean_expression THEN result_expression 子句。
- ☑ THEN result_expression：当 input_expression=when_expression 取值为 TRUE，或者 Boolean_expression 取值为 TRUE 时返回的表达式。result_expression 是任意有效的 SQL Server 表达式。
- ☑ ELSE else_result_expression：当比较运算取值不为 TRUE 时返回的表达式。如果省略此参数并且比较运算取值不为 TRUE，CASE 将返回 NULL 值。else_result_expression 是任意有效的 SQL Server 表达式。else_result_expression 和所有 result_expression 的数据类型必须相同，或者必须是隐性转换。

【例 17.04】　使用 CASE 语句实现交叉表查询。（**实例位置：资源包\源码\17\17.04**）

在 sp 表中，按照“商品名称”进行交叉表查询。结果表显示各商品各月的销售情况。SQL 语句如下：

```
USE   student
SELECT 月份,SUM(CASE 商品名称 WHEN '李小葱专辑' THEN 销售数量 ELSE NULL END)AS [李小葱专辑],SUM(CASE 商品名称 WHEN '周木人专辑' THEN 销售数量 ELSE NULL END)AS [周木人专辑] ,SUM(CASE 商品名称 WHEN '国产 E601' THEN 销售数量 ELSE NULL END)AS [E601],SUM(CASE 商品名称 WHEN '920 演唱会 DVD' THEN 销售数量 ELSE NULL END)AS [920 演唱会 DVD] FROM sp GROUP BY 月份
```

实现的结果如图 17.14 所示。

	月份	李小葱专辑	周木人专辑	E601	920演唱会DVD
1	9	888	777	NULL	NULL
2	10	28888	778	NULL	333
3	11	9999	NULL	564	15554
4	12	NULL	NULL	2478	NULL

图 17.14　sp 表按照商品名称交叉表查询

在 sp 表中，按照“月份”进行交叉表查询。SQL 语句如下：

```
USE student
SELECT 商品名称,SUM(CASE 月份 WHEN '9' THEN 销售数量 ELSE NULL END)AS [9 月份],SUM(CASE 月
```

```
份 WHEN '10' THEN 销售数量 ELSE NULL END) AS [10月份] ,SUM(CASE 月份 WHEN '11' THEN 销售数量
ELSE NULL END)AS [11月份],SUM(CASE 月份 WHEN '12' THEN 销售数量 ELSE NULL END)AS [12月份]
FROM sp GROUP BY 商品名称
```

实现的结果如图17.15所示。

	商品名称	9月份	10月份	11月份	12月份
1	920演唱会DVD	NULL	333	15554	NULL
2	国产E601	NULL	NULL	564	2478
3	李小葱专辑	888	28888	9999	NULL
4	周木人专辑	777	778	NULL	NULL

图17.15　sp表按照月份交叉表查询

17.3　小　　结

本章介绍了关于SQL Server 2014的高级应用，如用户自定义函数和交叉表查询。读者通过创建用户自定义函数可以实现将代码封装在一个函数体内方便调用；可以使用PIVOT、UNPIVOT运算符以及CASE语句实现交叉表查询。

第18章

SQL Server 安全管理

（视频讲解：21 分钟）

本章主要介绍 SQL Server 2014 安全管理，主要包括 SQL Server 身份验证、数据库用户、SQL Server 角色和管理 SQL Server 权限。通过本章的学习，读者能够使用 SQL Server 的安全管理工具构造灵活、安全的管理机制。

学习摘要：

- SQL Server 的登录验证模式
- 创建以 SQL Server 方式登录的登录名
- 更改登录名
- 使用 SQL 语句管理登录名
- 为用户设置访问权限


视频讲解

18.1　SQL Server 身份验证

18.1.1　验证模式

验证模式指数据库服务器如何处理用户名与密码。SQL Server 2014 的验证模式包括 Windows 验证模式与混合验证模式。

1．Windows 验证模式

Windows 验证模式是 SQL Server 2014 使用 Windows 操作系统中的信息验证账户名和密码。这是默认的身份验证模式，比混合模式安全。Windows 验证使用 Kerberos 安全协议，通过强密码的复杂性验证提供密码策略强制，提供账户锁定与密码过期功能。

2．混合模式

允许用户使用 Windows 身份验证或 SQL Server 身份验证进行连接。通过 Windows 用户账户连接的用户可以使用 Windows 验证的受信任连接。

18.1.2　配置 SQL Server 的身份验证模式

SQL Server 2014 的验证方式可以通过 SQL Server Management Studio 工具进行设置。具体设置步骤如下。

（1）通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单打开 SQL Server Management Studio 工具。

（2）打开 SQL Server Management Studio 后，弹出“连接到数据库引擎”对话框。输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，如图 18.1 所示，单击“连接”按钮连接到服务器中。



图 18.1　“连接到数据库引擎”对话框

（3）服务器连接完成后，用鼠标右键单击“对象资源管理器”中的服务器，在弹出的快捷菜单中

选择“属性”命令，如图 18.2 所示。

（4）弹出“服务器属性”窗口，打开“安全性”页面，如图 18.3 所示。

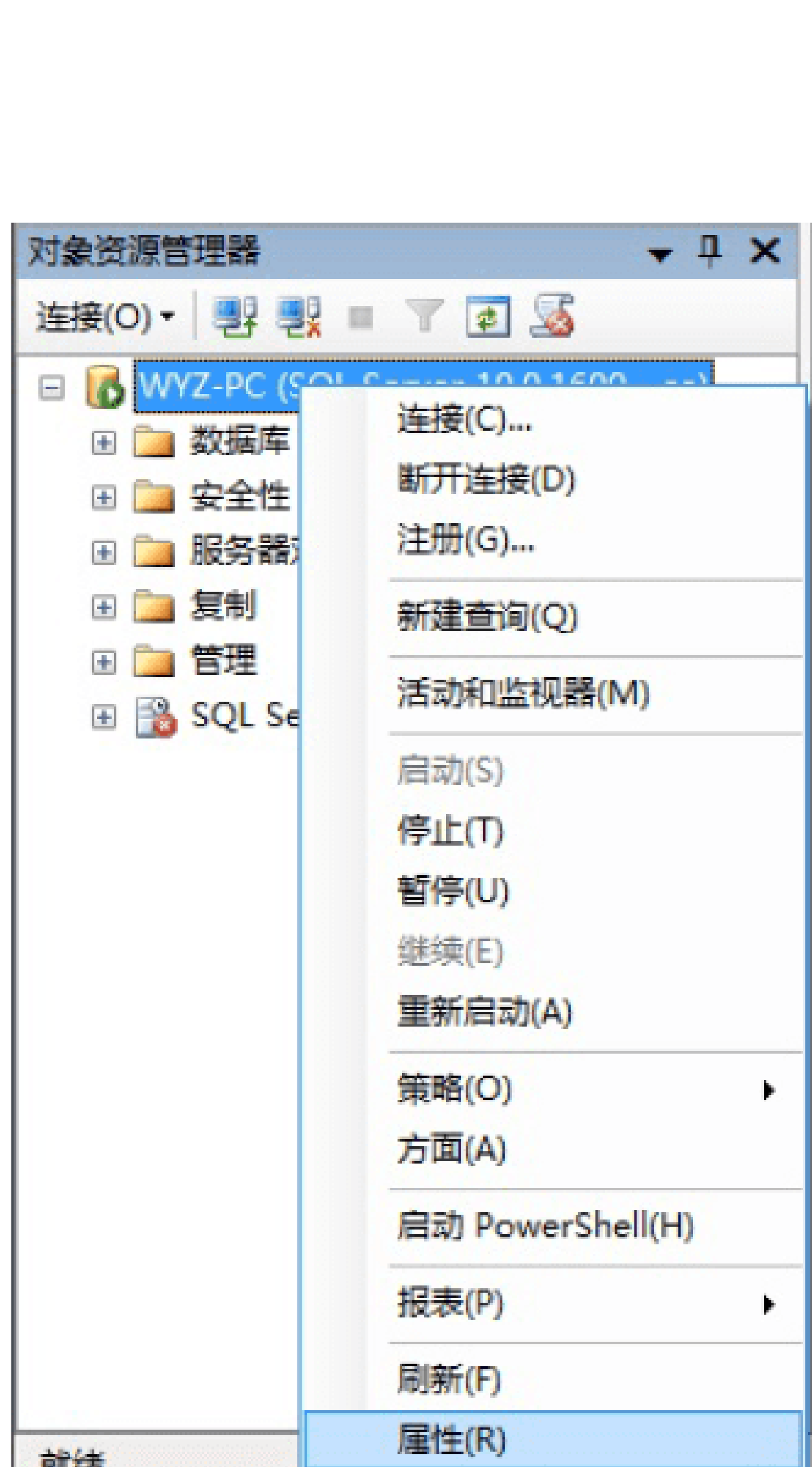


图 18.2　选择命令



图 18.3　“服务器属性”窗口

（5）在“服务器属性”窗口的“安全性”页面中设置 SQL Server 的验证模式。单击“确定”按钮，即可更改验证模式。

18.1.3　管理登录账号

在 SQL Server 2014 中有两个登录账户：一个是登录服务器的登录名；另外一个是使用数据库的用户账号。登录名是指能登录到 SQL Server 的账号，它属于服务器的层面，本身并不能让用户访问服务器中的数据库，而登录者要使用服务器中的数据库时，必须要有用户账号才能存取数据库。本节介绍如何创建、修改和删除 SQL Server 登录名。

管理员可以通过 SQL Server Management Studio 工具对 SQL Server 2014 中的登录名进行创建、修改、删除等管理。

1．创建登录名

创建登录名可以通过手动创建或执行 SQL 语句实现，手动创建登录名要比执行 SQL 语句创建更直观、简单，建议初学 SQL Server 的人员采用该种方法。下面分别使用这两种方法创建登录名，具体步骤如下。

（1）手动创建登录名

① 通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

② 在弹出的“连接到数据库引擎”对话框，输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

③ 单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“安全性”→“登录名”，并在“登录名”上单击鼠标右键，在弹出的快捷菜单中选择“新建登录名”命令，如图 18.4 所示。



图 18.4 “新建登录名”命令

④ 打开“登录名-新建”窗口，如图 18.5 所示。

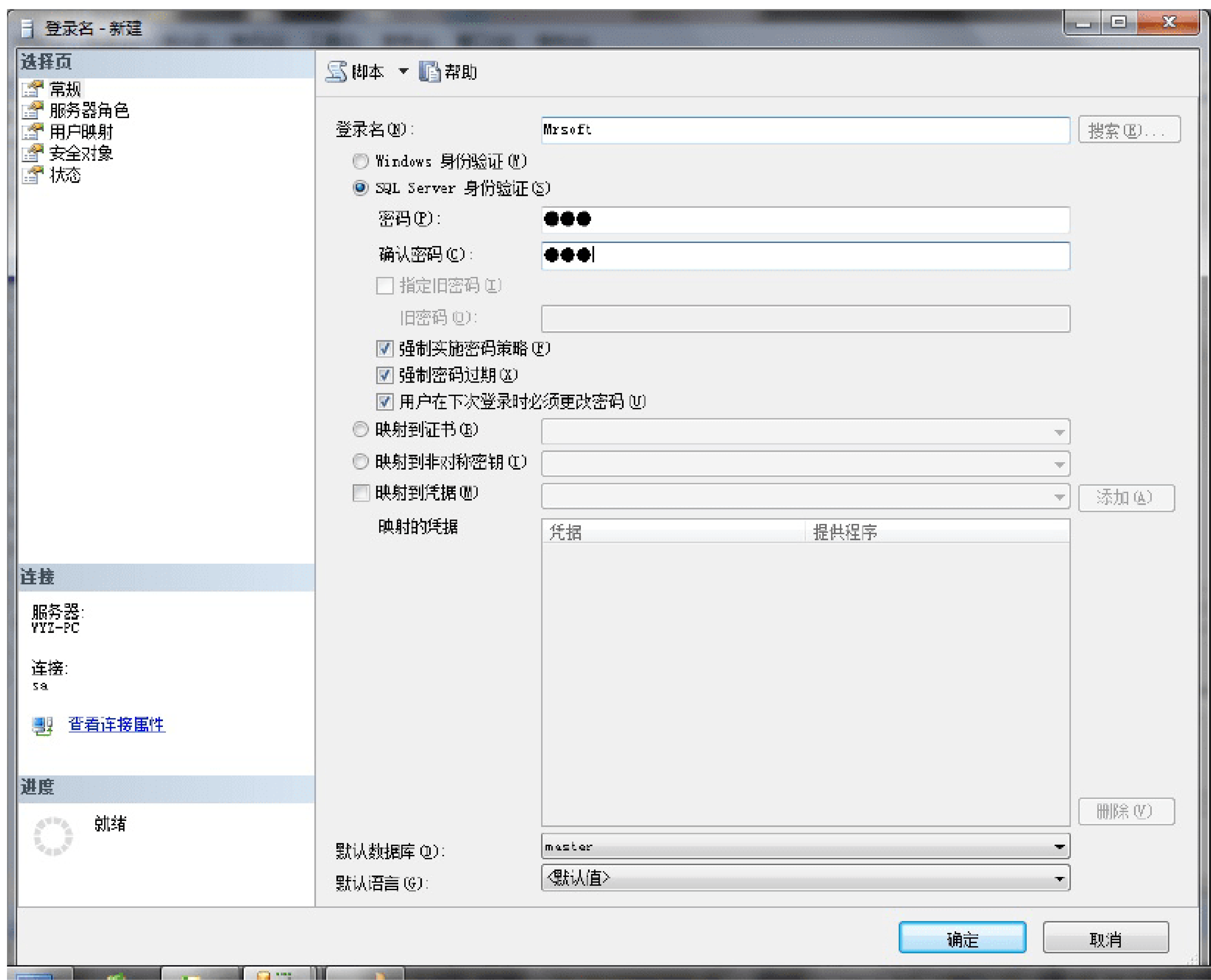


图 18.5 “登录名-新建”窗口

⑤ 在“登录名”文本框中输入所创建登录名的名称。若选中“Windows 身份验证”单选按钮，可通过单击“搜索”按钮，查找并添加 Windows 操作系统中的用户名称；若选中“SQL Server 身份验证”单选按钮，则需在“密码”与“确认密码”文本框中输入登录时采用的密码。

⑥ 在“默认数据库”与“默认语言”下拉列表框中选择该登录名登录 SQL Server 2014 后默认使用的数据库与语言。

⑦ 单击“确定”按钮，即可完成创建 SQL Server 登录名。

（2）执行 SQL 语句创建登录名

在 SQL Server Management Studio 工具中也可通过执行 CREATE LOGIN 语句创建登录名。语法格式如下：

```
CREATE LOGIN login_name
  {
    WITH
      <
        PASSWORD = 'password'
        [HASHED]
        [MUST_CHANGE]
        [
          ,
          <
            SID = sid
            |
            DEFAULT_DATABASE = database
            |
            DEFAULT_LANGUAGE = language
            |
            CHECK_EXPIRATION = {ON | OFF}
            |
            CHECK_POLICY = {ON | OFF}
            [CREDENTIAL = credential_name]
          >
          [,...]
        ]
      >
    |
    FROM
    <
      WINDOWS
        [
          WITH
            <
              DEFAULT_DATABASE = database
              |
              DEFAULT_LANGUAGE = language
            >
          [,...]
        ]
      |
      CERTIFICATE certname
      |
      ASYMMETRIC KEY asym_key_name
      >
 }
```

参数的说明如表 18.1 所示。

表 18.1　CREATE LOGIN 语句语法中参数的说明

参　　数	说　　明
login_name	指定创建的登录名。有 4 种类型的登录名：SQL Server 登录名、Windows 登录名、证书映射登录名和非对称密钥映射登录名。如果从 Windows 域账户映射 login_name，则 login_name 必须用方括号（[]）括起来
PASSWORD = 'password'	仅适用于 SQL Server 登录名。指定正在创建的登录名的密码。此值提供时可能已经过哈希运算
HASHED	仅适用于 SQL Server 登录名。指定在 PASSWORD 参数后输入的密码已经过哈希运算。如果未选择此选项，则在将作为密码输入的字符串存储到数据库之前，对其进行哈希运算
MUST_CHANGE	仅适用于 SQL Server 登录名。如果包括此选项，则 SQL Server 将在首次使用新登录名时提示用户输入新密码
SID = sid	仅适用于 SQL Server 登录名。指定新 SQL Server 登录名的 GUID。如果未选择此选项，则 SQL Server 将自动指派 GUID
DEFAULT_DATABASE = database	指定将指派给登录名的默认数据库。默认设置为 master 数据库
DEFAULT_LANGUAGE = language	指定将指派给登录名的默认语言，默认语言设置为服务器的当前默认语言。即使服务器的默认语言发生更改，登录名的默认语言仍保持不变
CHECK_EXPIRATION = {ON | OFF}	仅适用于 SQL Server 登录名。指定是否对此登录名强制实施密码过期策略。默认值为 OFF
CHECK_POLICY = {ON | OFF}	仅适用于 SQL Server 登录名。指定应对此登录名强制实施运行 SQL Server 的计算机的 Windows 密码策略。默认值为 ON
CREDENTIAL = credential_name	将映射到新 SQL Server 登录名的凭据名称。该凭据必须已存在于服务器中
WINDOWS	指定将登录名映射到 Windows 登录名
CERTIFICATE certname	指定将与此登录名关联的证书名称。此证书必须已存在于 master 数据库中
ASYMMETRIC KEY asym_key_name	指定将与此登录名关联的非对称密钥的名称。此密钥必须已存在于 master 数据库中

例如，使用该语句创建以 SQL Server 方式登录的登录名，代码如下：

```
CREATE LOGIN Mr WITH PASSWORD = 'MrSoft'
```

执行 SQL 语句创建登录名具体步骤如下。

① 通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

② 在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

③ 单击工具栏中的新建查询(N)按钮，打开查询编辑器窗口。该窗口可以用来创建和运行 Transact-SQL 脚本，如图 18.6 所示。

④ 在查询编辑器窗口内编辑创建登录名的 SQL 语句。按 F5 键执行编辑的 SQL 语句，完成创建登录名操作，如图 18.7 所示。

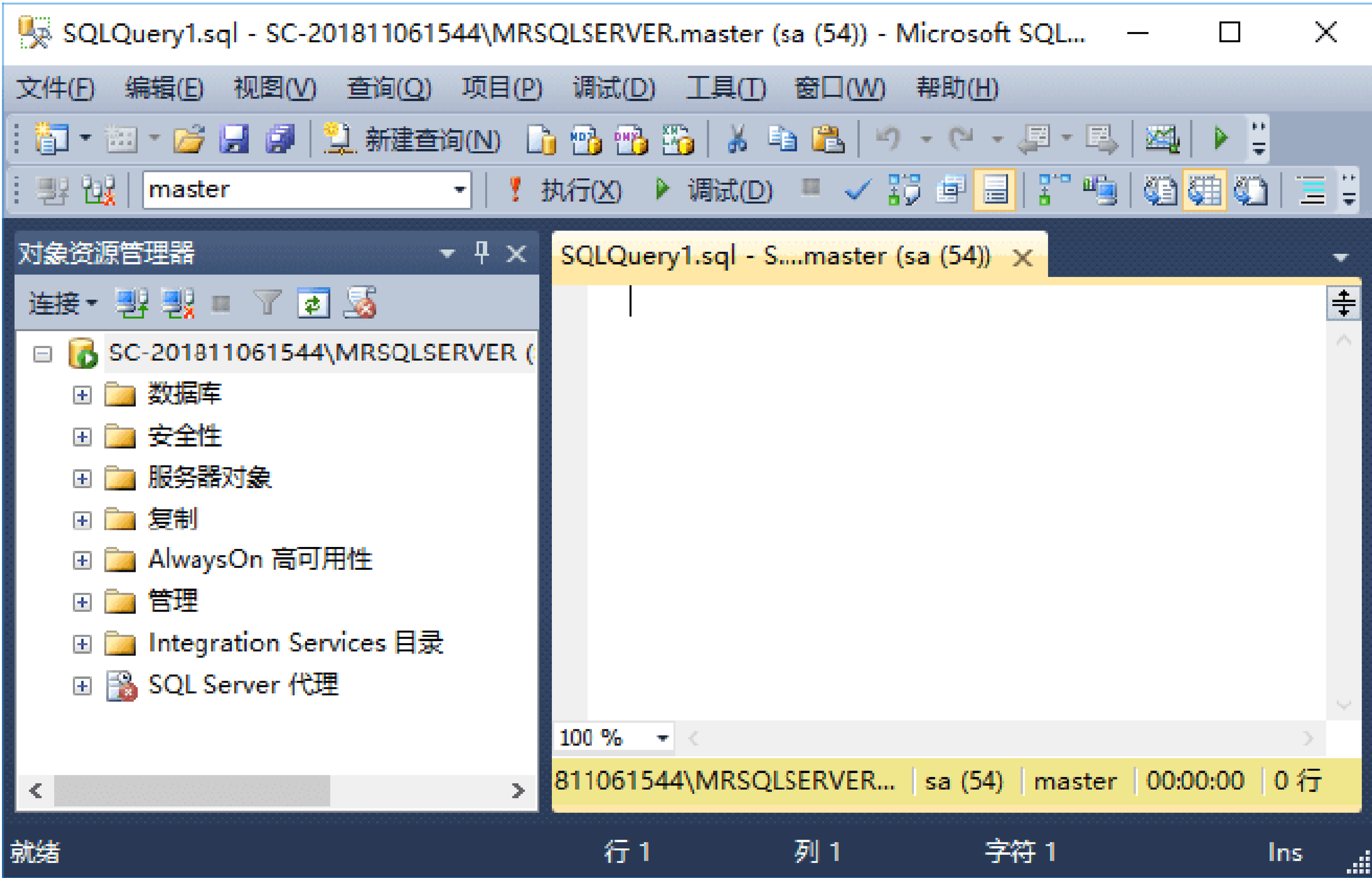


图 18.6　查询编辑器窗口

2．修改登录名

（1）手动修改登录名

① 在“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

② 在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

③ 单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“安全性”→“登录名”。

④ 选择“登录名”下需要修改的登录名，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，如图 18.8 所示。

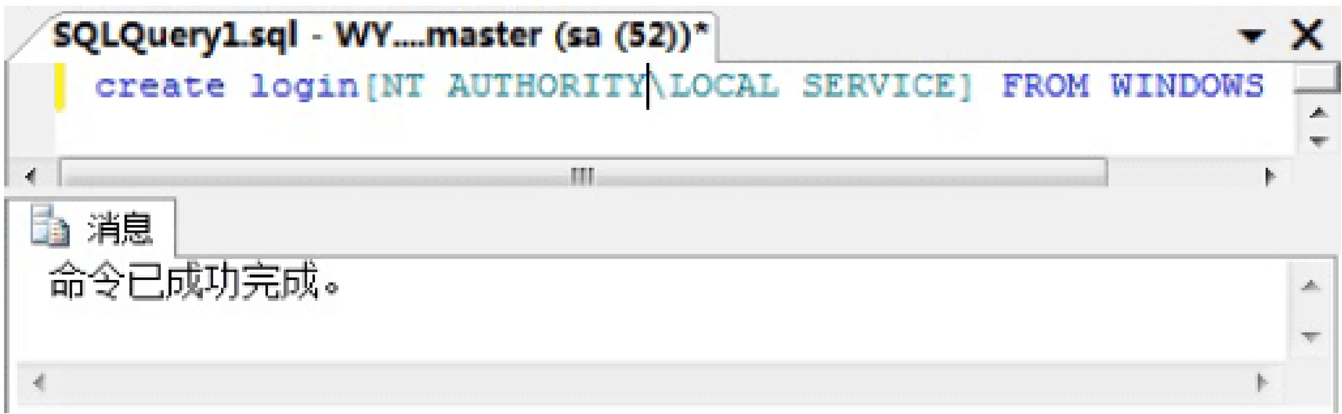


图 18.7　执行 SQL 语句创建登录名

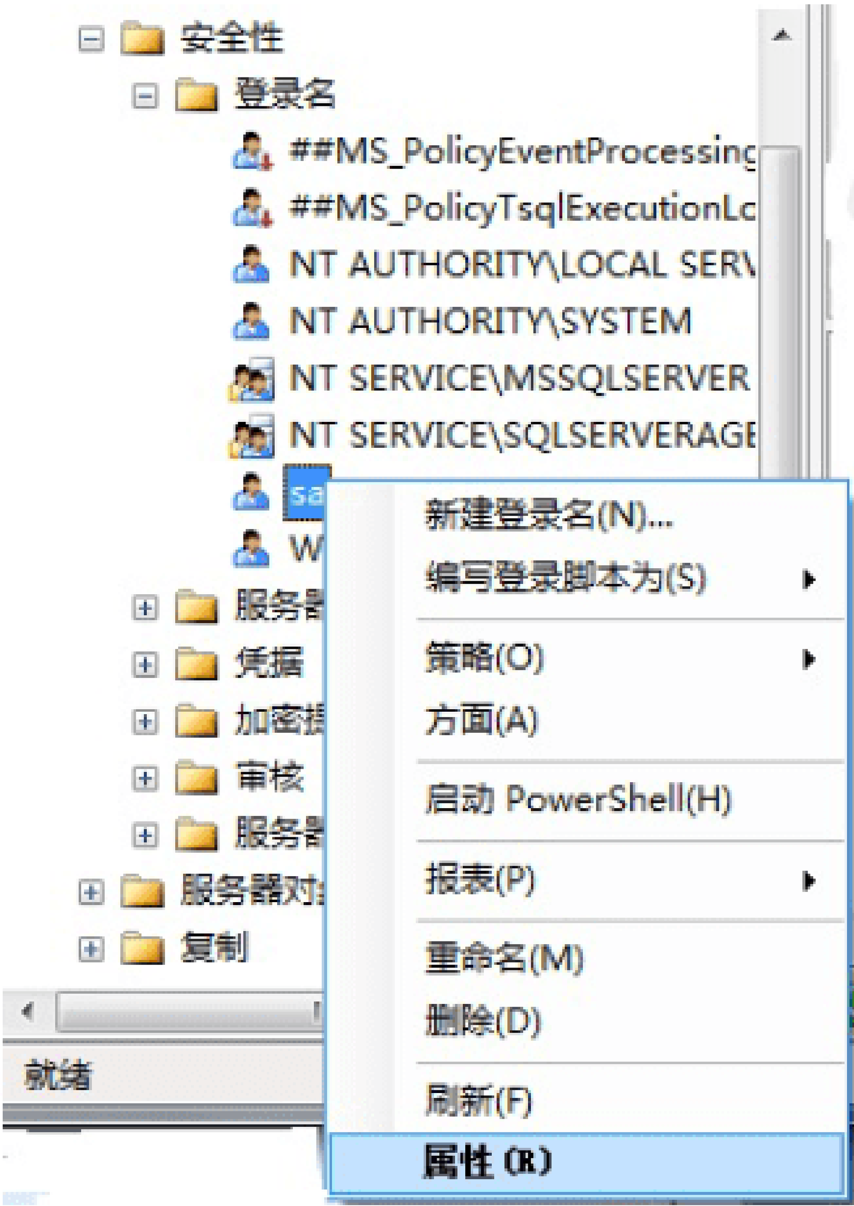


图 18.8　修改登录名

⑤ 在弹出的“登录属性”窗口中修改有关该登录名的信息，如图 18.9 所示，单击“确定”按钮即可完成修改。

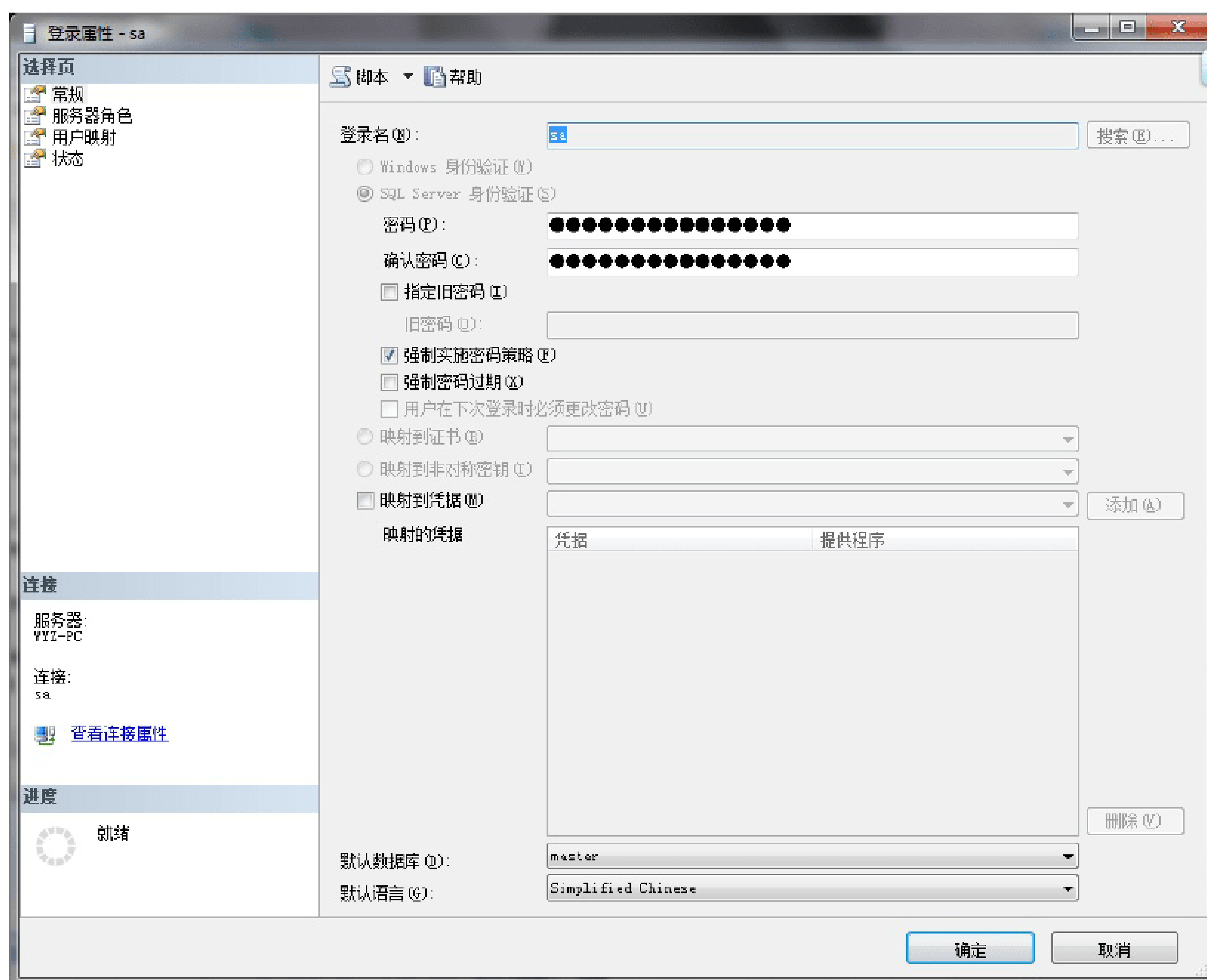


图 18.9 “登录属性”窗口

（2）执行 SQL 语句修改登录名

通过执行 ALTER LOGIN 语句，也可以修改更改 SQL Server 登录名的属性。语法格式如下：

```
ALTER LOGIN login_name
  {
    <
      ENABLE | DISABLE
    >
    |
    WITH
      <
        PASSWORD = 'password'
        [
          OLD_PASSWORD = 'oldpassword'
          | <MUST_CHANGE | UNLOCK>
          [<MUST_CHANGE | UNLOCK>]
       ]
        | DEFAULT_DATABASE = database
        | DEFAULT_LANGUAGE = language
        | NAME = login_name
        | CHECK_POLICY = {ON | OFF}
        | CHECK_EXPIRATION = {ON | OFF}
        | CREDENTIAL = credential_name
        | NO CREDENTIAL
```

```
    >
    [,...]
}
```

参数的说明如表 18.2 所示。

表 18.2　ALTER LOGIN 语句语法参数的说明

参　　数	说　　明
login_name	指定正在更改的 SQL Server 登录的名称
ENABLE | DISABLE	启用或禁用此登录
PASSWORD = 'password'	仅适用于 SQL Server 登录账户。指定正在更改的登录的密码
OLD_PASSWORD = 'oldpassword'	仅适用于 SQL Server 登录账户。要指派新密码登录的当前密码
MUST_CHANGE	仅适用于 SQL Server 登录账户。如果包括此选项，则 SQL Server 将在首次使用已更改的登录时提示输入更新的密码
UNLOCK	仅适用于 SQL Server 登录账户。指定应解锁被锁定的登录
DEFAULT_DATABASE = database	指定将指派给登录的默认数据库
DEFAULT_LANGUAGE = language	指定将指派给登录的默认语言
NAME = login_name	正在重命名的登录的新名称。如果是 Windows 登录，则与新名称对应的 Windows 主体的 SID 必须匹配与 SQL Server 中的登录相关联的 SID。SQL Server 登录的新名称不能包含反斜杠字符（\）
CHECK_POLICY = {ON | OFF}	仅适用于 SQL Server 登录账户。指定应对此登录账户强制实施运行 SQL Server 的计算机的 Windows 密码策略。默认值为 ON
CHECK_EXPIRATION = {ON | OFF}	仅适用于 SQL Server 登录账户。指定是否对此登录账户强制实施密码过期策略。默认值为 OFF
CREDENTIAL = credential_name	将映射到 SQL Server 登录的凭据的名称。该凭据必须已存在于服务器中
NO CREDENTIAL	删除登录到服务器凭据的当前所有映射

例如，使用该语句更改 SQL Server 登录方式的登录名密码，代码如下：

```
ALTER LOGIN sa WITH PASSWORD = ''
```

执行 SQL 语句修改登录名属性的具体步骤如下。

① 通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

② 在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

③ 单击工具栏中的“新建查询(N)”按钮，打开查询编辑器窗口。

④ 在查询编辑器窗口内编辑修改登录名的 SQL 语句。按 F5 键执行编辑的 SQL 语句，完成修改登录名的操作，如图 18.10 所示。

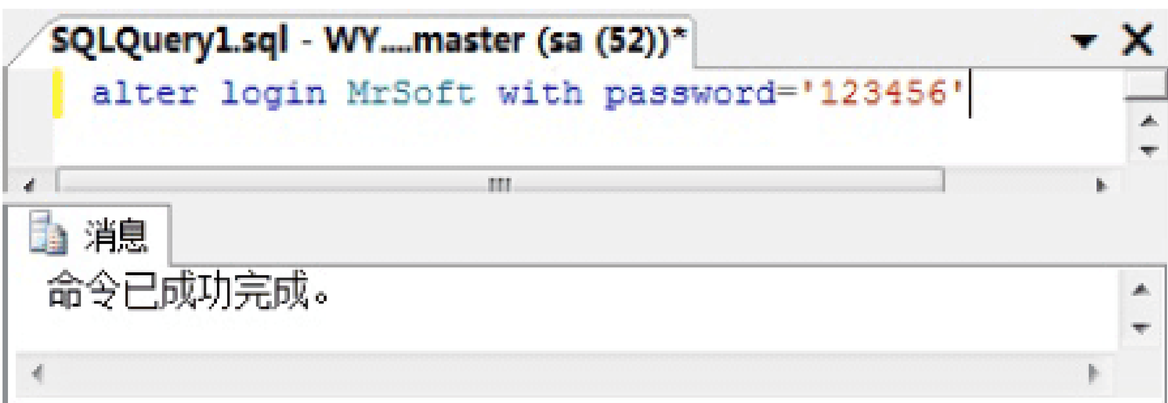


图 18.10　执行 SQL 语句修改登录名属性

3．删除登录名

当 SQL Server 2014 中的登录名不再使用时，就可以将

其删除。与创建、修改登录名相同，删除登录名也可以通过手动及执行 SQL 语句来实现。

（1）手动删除登录名

① 通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

② 在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

③ 单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“安全性”→“登录名”。

④ 选择“登录名”下需要修改的登录名，单击鼠标右键，在弹出的快捷菜单中选择“删除”命令，如图 18.11 所示。

⑤ 打开“删除对象”窗口，在该窗口中确认删除的登录名。确认后单击“确定”按钮，将该登录名删除，如图 18.12 所示。

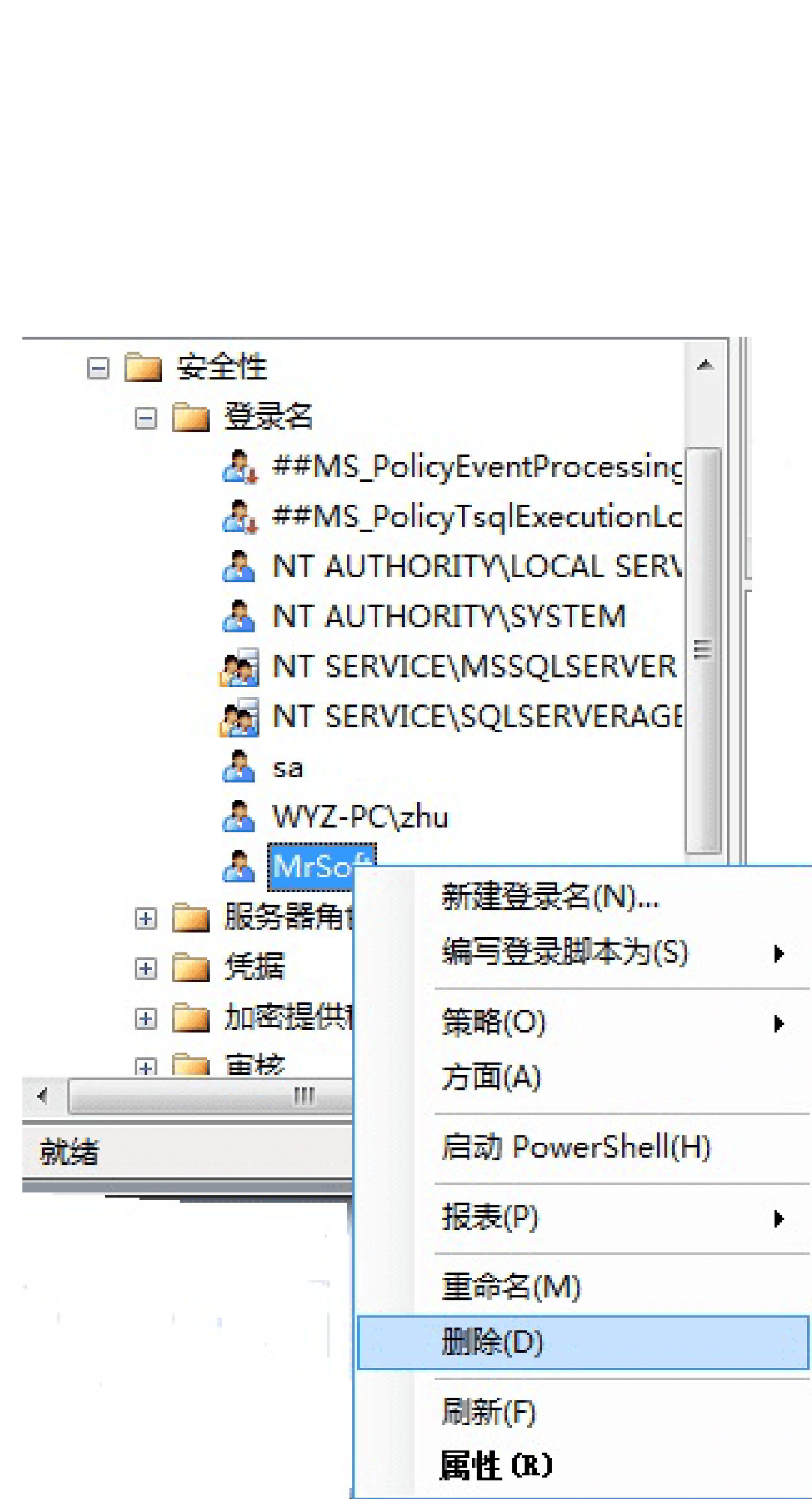


图 18.11　选择“删除”命令

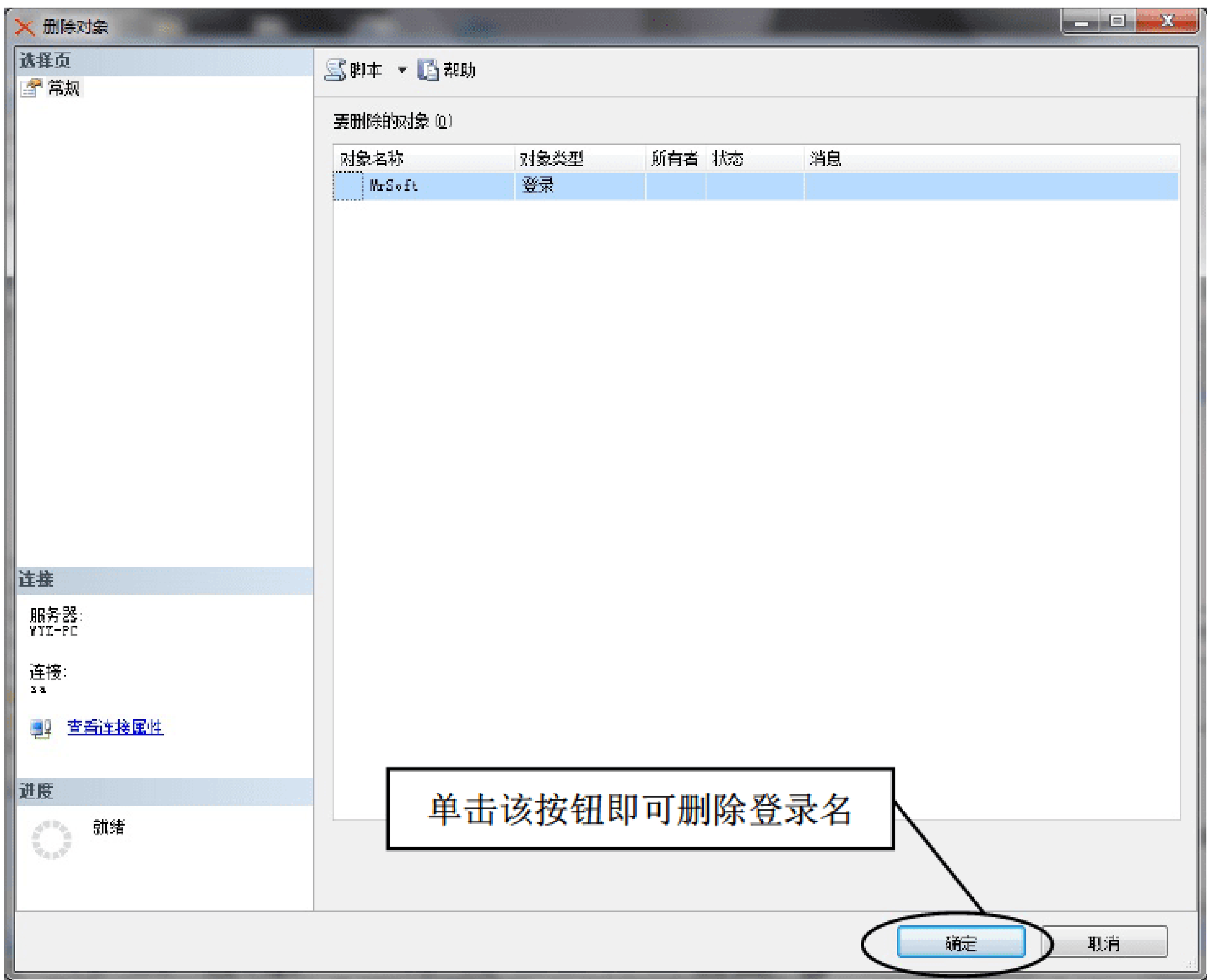


图 18.12　“删除对象”窗口

（2）执行 SQL 语句删除登录名

通过执行 DROP LOGIN 语句可以将 SQL Server 2014 中的登录名。语法格式如下：

```
DROP LOGIN login_name
```

login_name 为指定要删除的登录名。

例如，使用该语句删除 MrSoft 登录名，代码如下：

```
DROP LOGIN MrSoft
```

执行 SQL 语句删除登录名具体步骤如下。

① 通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

② 在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

③ 单击工具栏中的新建查询(N)按钮，打开查询编辑器窗口。

④ 在查询编辑器窗口内编辑删除登录名的 SQL 语句。按 F5 键执行编辑的 SQL 语句，完成删除登录名的操作，如图 18.13 所示。

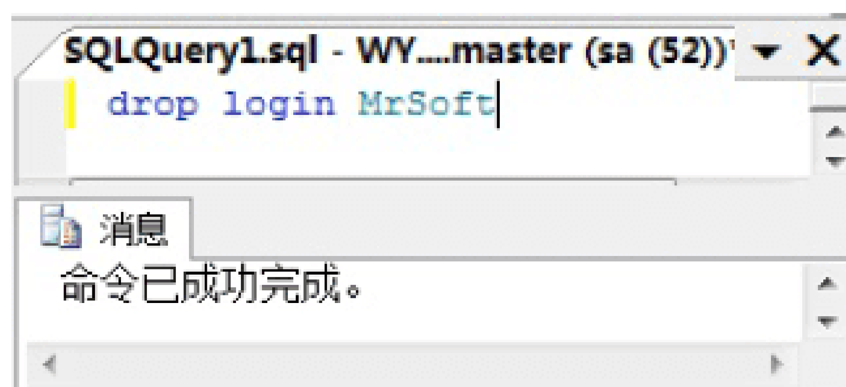
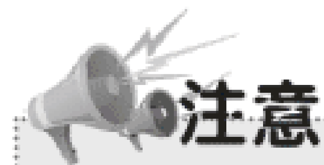


图 18.13　执行 SQL 语句删除登录名



18.2　数据库用户

登录名创建之后，用户只能通过该登录名访问整个 SQL Server 2014，而不是 SQL Server 2014 中的某个数据库。若要使用户能够访问 SQL Server 2014 中的某个数据库，还需要给这个用户授予访问某个数据库的权限，也就是在所要访问的数据库中为该用户创建一个数据库用户账户。



注意

默认情况下，数据库创建时就包含一个 guest 用户。guest 用户不能删除，但可以通过在除 master 和 temp 以外的任何数据库中执行 REVOKECONNECT FROM GUEST 来禁用该用户。

18.2.1　创建数据库用户

创建数据库用户的具体步骤如下。

（1）通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“数据库”→“数据库名称”→“安全性”→“用户”，并在“用户”上单击鼠标右键，在弹出的快捷菜单中选择“新建用户”命令，如图 18.14 所示。

（4）打开“数据库用户”窗口，通过该窗口输入要创建的用户名，并选择使用的登录名。设置该用户拥有的架构与数据库角色成员。单击“确定”按钮即可创建该用户。“数据库用户”窗口如图 18.15 所示。

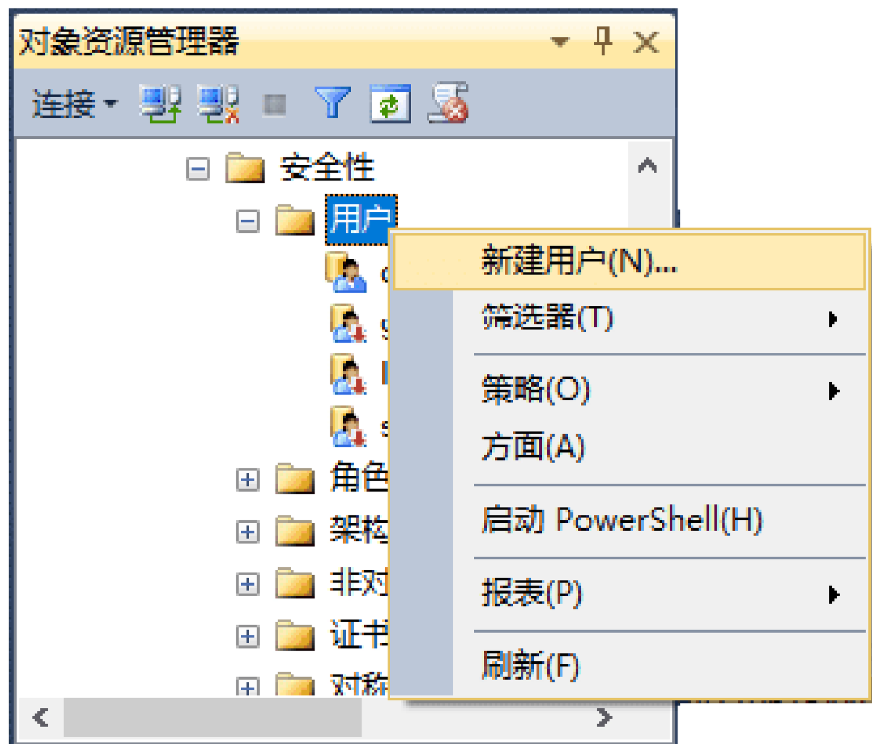


图 18.14 选择“新建用户”命令

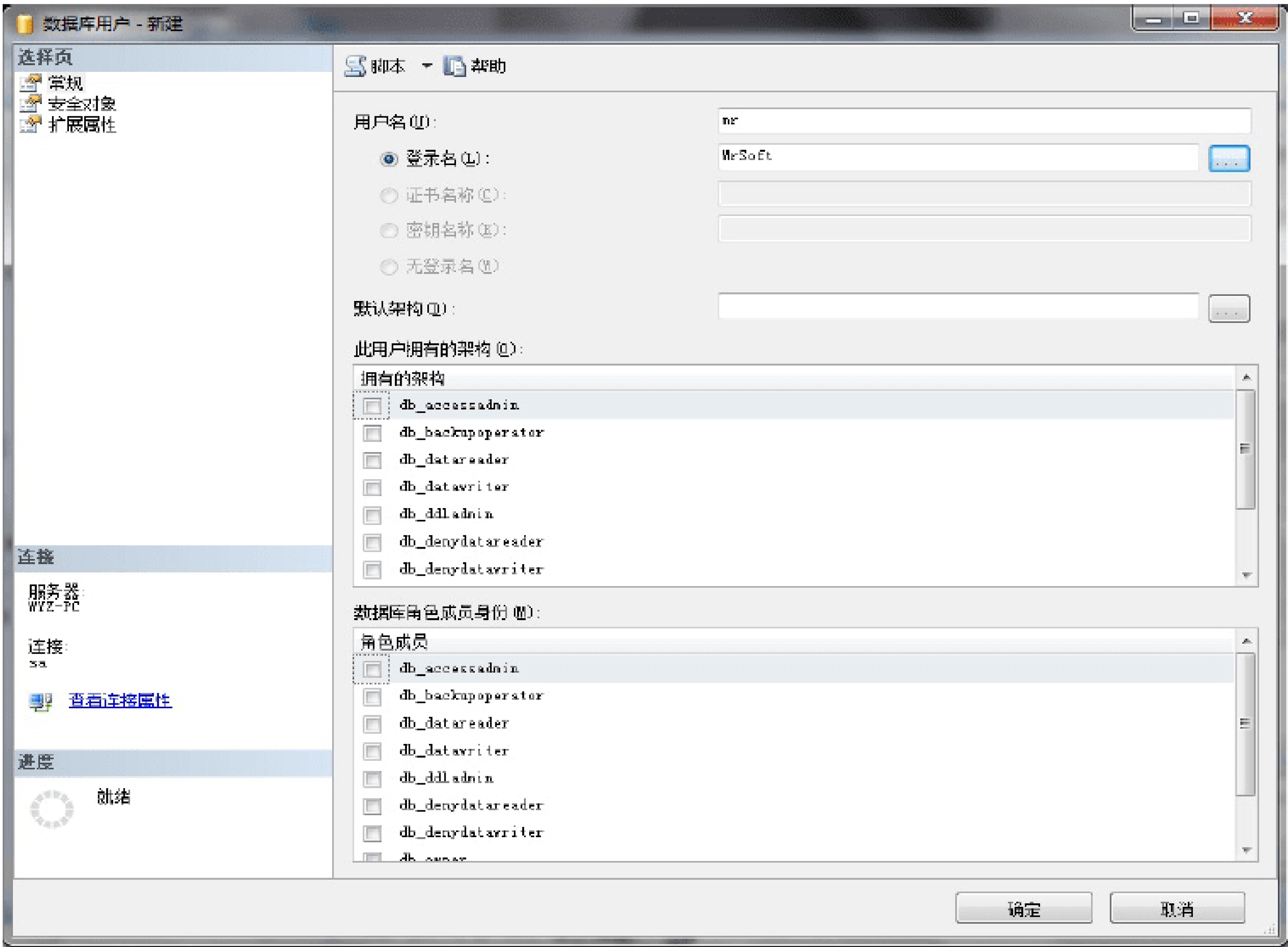


图 18.15 “数据库用户”窗口

18.2.2 删除数据库用户

删除数据库用户具体步骤如下。

（1）通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“数据库”→“数据库名称”→“安全性”→“用户”，在要删除的用户上单击鼠标右键，如 mr，在弹出的快捷菜单中选择“删除”命令，如图 18.16 所示。

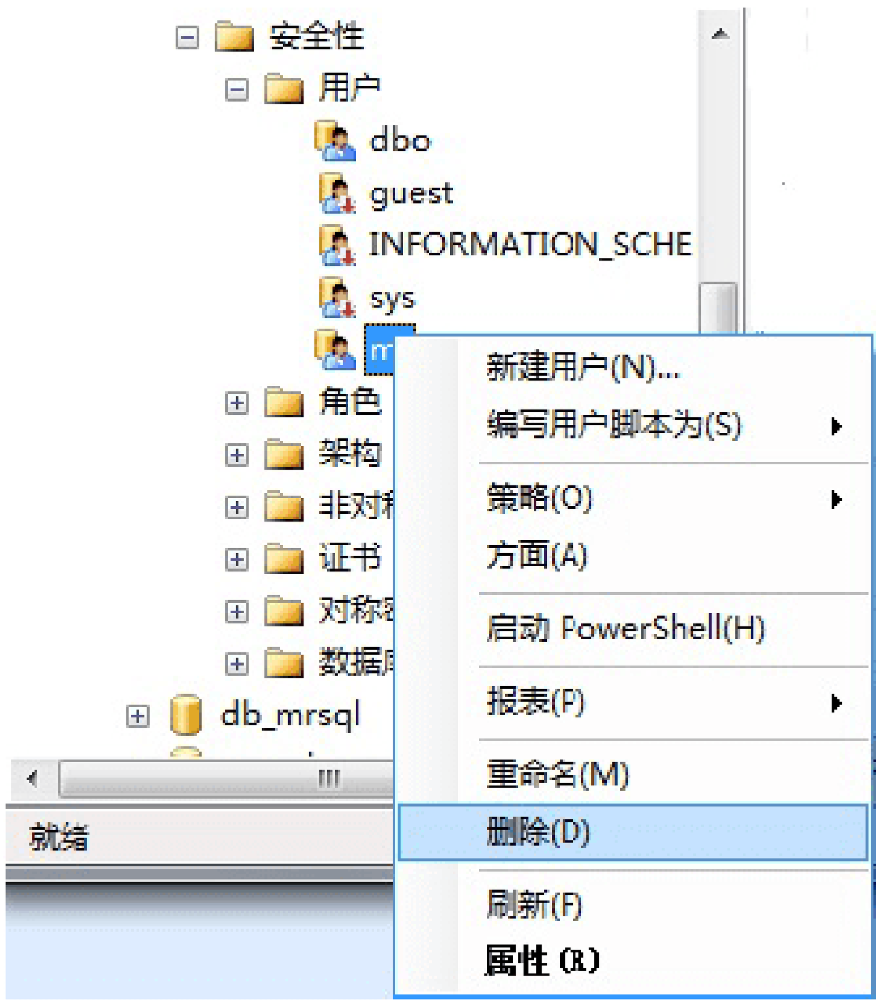


图 18.16 删除用户

（4）在弹出的“删除对象”窗口中确认删除的用户名称，单击“确定”按钮即可将该用户删除。


视频讲解

18.3　SQL Server 角色

角色是指用户对 SQL Server 进行的操作类型。角色根据权限的划分可以分为固定服务器角色与固定数据库角色。

18.3.1　固定服务器角色

SQL Server 自动在服务器级别预定义了固定服务器角色与相应的权限，如表 18.3 所示。

表 18.3　固定服务器角色与相应的权限

固定服务器角色名称	权　　限
bulkadmin	该角色可以运行 BULK INSERT 语句
dbcreator	该角色可以创建、更改、删除和还原任何数据库
diskadmin	该角色用于管理磁盘文件
processadmin	该角色可以终止 SQL Server 实例中运行的进程（结束进程）
securityadmin	该角色管理登录名及其属性（如分配权限、重置 SQL Server 登录名的密码）
serveradmin	该角色可以更改服务器范围的配置选项和关闭服务器
setupadmin	该角色可以管理以链接的服务器（如添加和删除链接服务器），并且也可以执行系统存储过程
sysadmin	该角色可以在服务器中执行任何操作。Windows BUILTIN\Administrators 组（本地管理员组）的所有成员都是 sysadmin 固定服务器角色的成员

18.3.2　固定数据库角色

固定数据库角色与相应的权限，如表 18.4 所示。

表 18.4　固定数据库角色与相应的权限

固定数据库角色名称	数据库级权限
db_accessadmin	该角色可以为 Windows 登录账户、Windows 组和 SQL Server 登录账户设置访问权限
db_backupoperator	该角色可以备份该数据库
db_datareader	该角色可以读取所有用户表中的所有数据
db_datawriter	该角色可以在所有用户表中添加、删除或更改数据
db_ddladmin	该角色可以在数据库中运行任何数据定义语言（DDL）命令
db_denydatareader	该角色不能读取数据库中用户表的任何数据
db_denydatawriter	该角色不能在数据库内的用户表中添加、修改或删除任何数据
db_owner	该角色可以执行数据库的所有配置和维护活动

续表

固定数据库角色名称	数据库级权限
db_securityadmin	该角色可以修改角色成员身份和管理权限
public	每个数据库用户都属于 public 数据库角色。当尚未对某个用户授予特定权限或角色时，则该用户将继承 public 角色的权限

18.3.3　管理 SQL Server 角色

为角色添加与删除用户，分为服务器角色与数据库角色两种，这两种的操作方法大致相同。下面分别介绍为服务器角色添加、删除用户与为数据库角色添加、删除用户的操作步骤。

1．为服务器角色添加、删除用户

（1）通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“安全性”→“服务器角色”，在“服务器角色”中选择需要设置的角色，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，如图 18.17 所示。

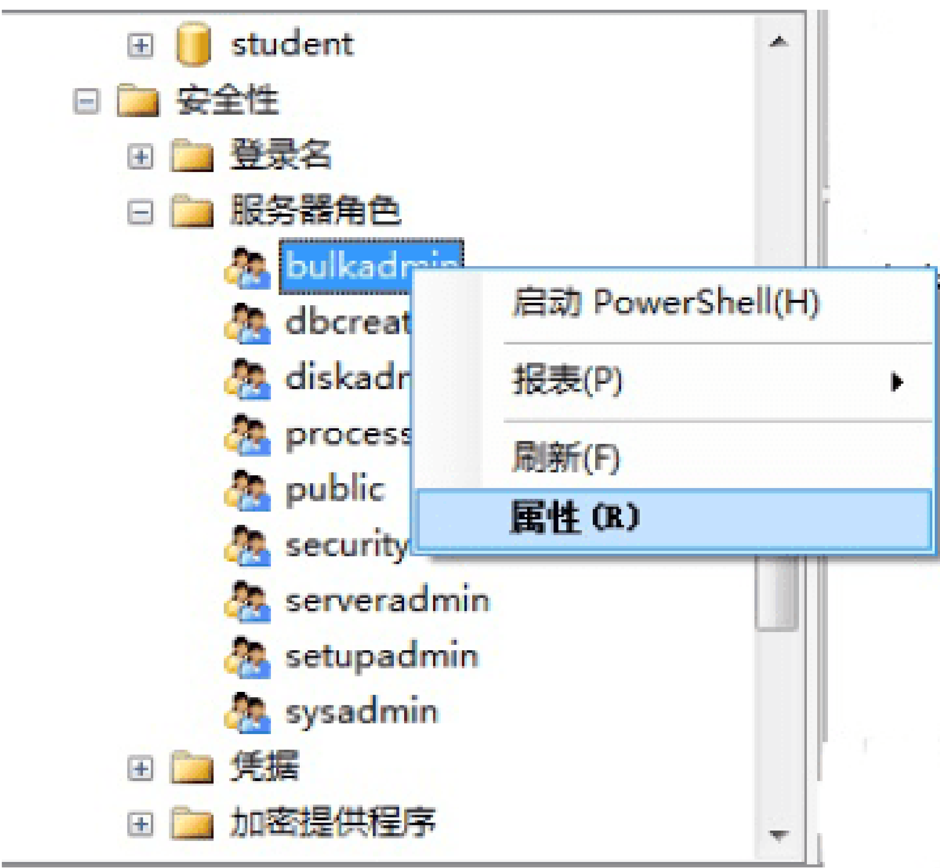


图 18.17　选择“属性”命令

（4）打开“服务器角色属性”窗口，单击“添加”按钮为服务器角色添加用户成员，单击“删除”按钮可以将选中的用户从该角色中删除。单击“确定”按钮即可完成对服务器角色所做的修改，如图 18.18 所示。

2．为数据库角色添加、删除用户

（1）通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

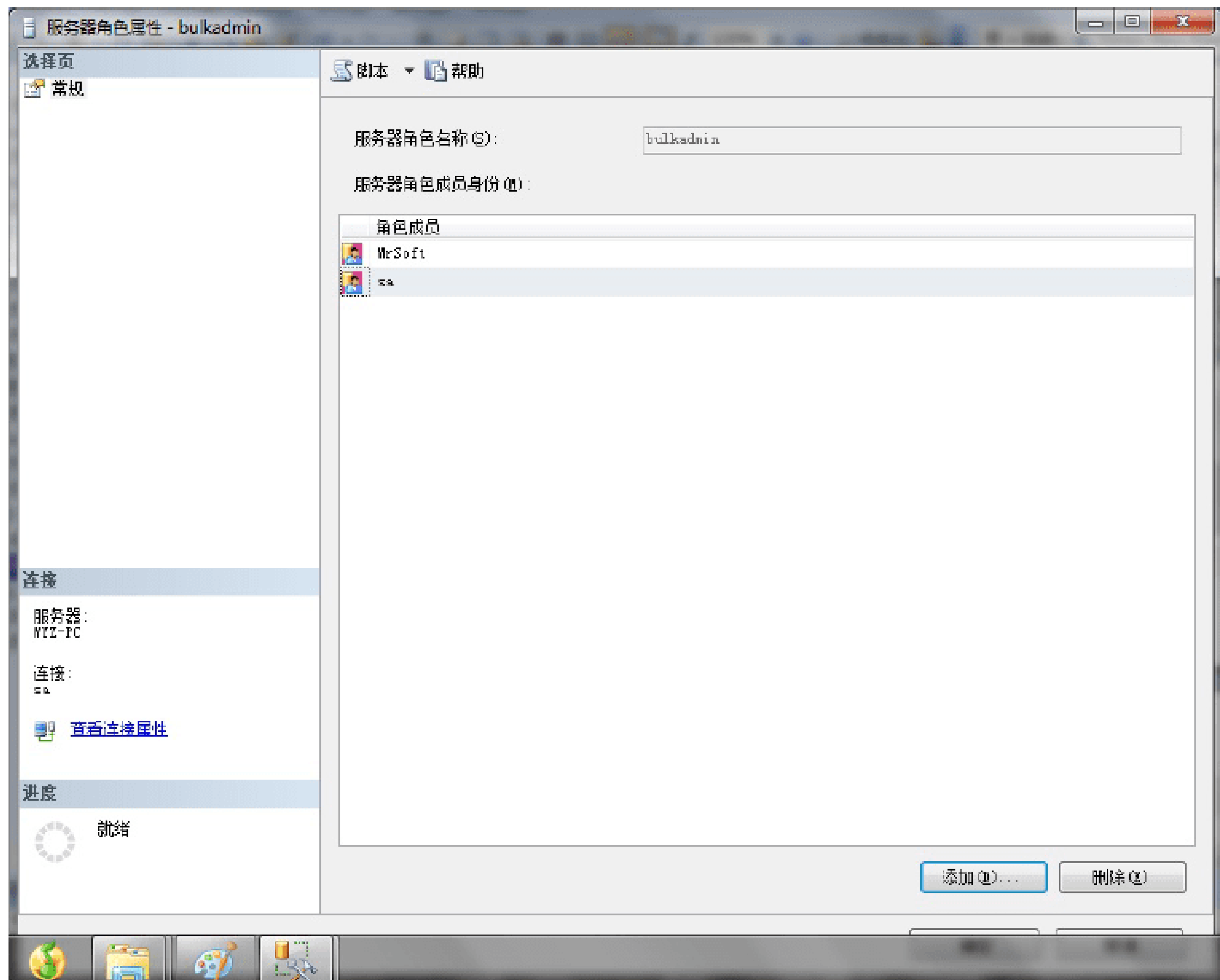


图 18.18　“服务器角色属性”窗口

（2）在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“数据库”→“数据库名称”→“安全性”→“角色”→“数据库角色”，在“数据库角色”中选择需要设置的角色，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令。

（4）打开“数据库角色属性”窗口，单击“添加”按钮为数据库角色添加用户成员，单击“删除”按钮可以将选中的用户从该角色中删除。单击“确定”按钮即可完成对数据库角色所做的修改。

18.4　管理 SQL Server 权限



视频讲解

权限用来控制用户对数据库访问与操作，可以通过 SQL Server Management Studio 工具对数据库中用户授予或删除访问与操作数据库的权限。

1. 授予权限

授予用户权限具体操作步骤如下。

（1）通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“数据库”→“数据库名称”→“安全性”→“用户”，在“用户”上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令。

（4）打开“数据库用户”窗口，在“选择页”中单击“安全对象”，如图 18.19 所示。

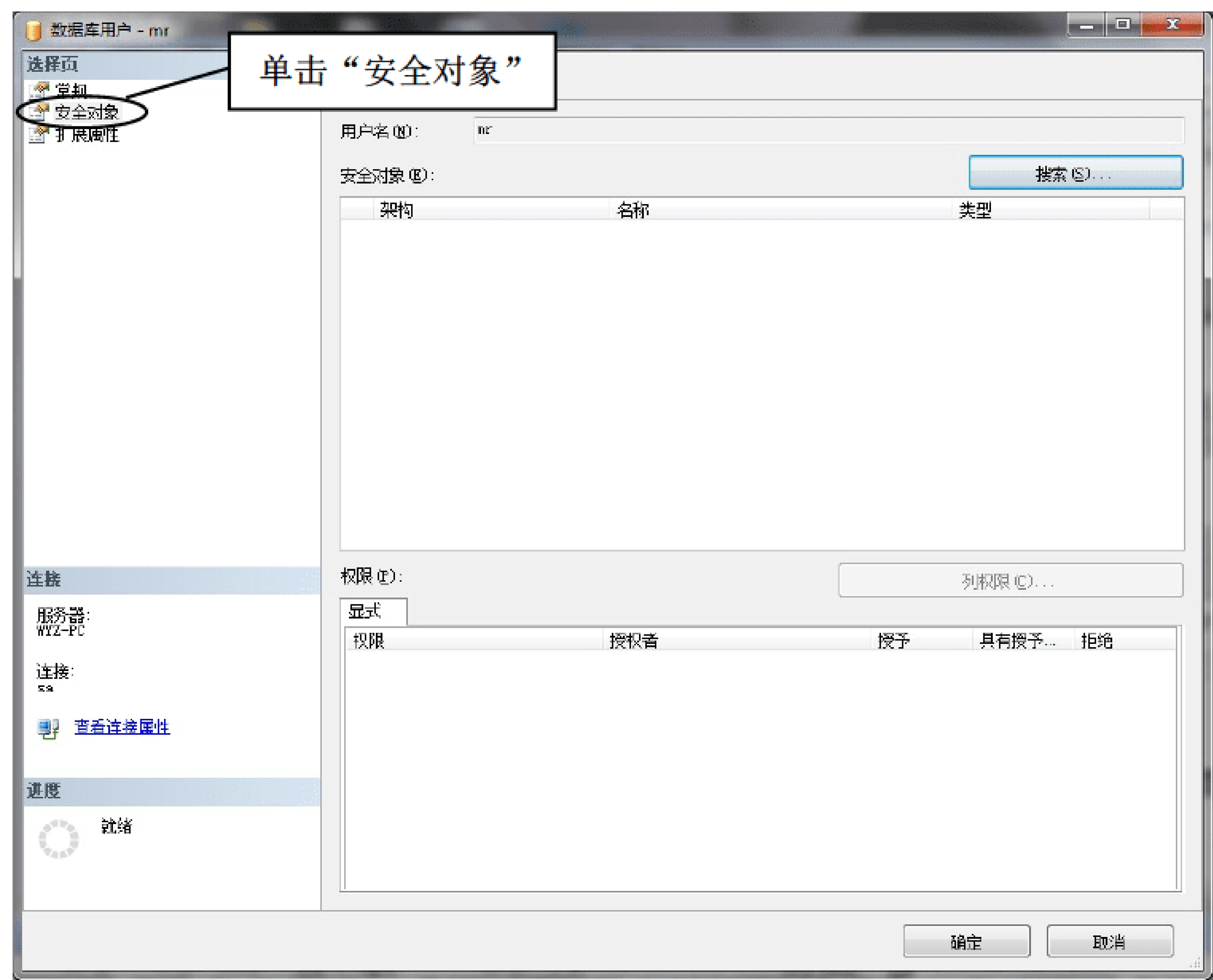


图 18.19　“数据库用户”窗口

（5）单击“添加”按钮，弹出“添加对象”对话框，通过该对话框选择对象类型限制。这里选中“特定类型的所有对象”单选按钮，如图 18.20 所示，单击“确定”按钮。

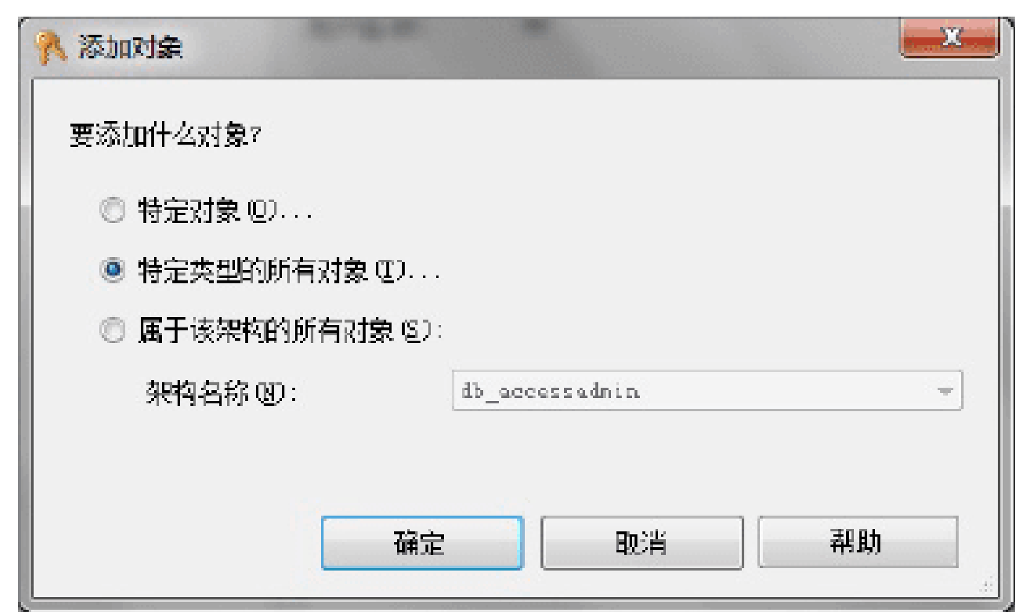


图 18.20　“添加对象”对话框



根据设置不同的操作，选择不同的对象。“特定对象”可以进一步定义对象搜索；“特定类型的所有对象”可以指定应包含在基础列表中的对象类型；“属于该架构的所有对象”用于添加到“架构名称”文本框中指定架构拥有的所有对象。

（6）打开“选择对象类型”对话框，如图 18.21 所示，在此选择访问及操作的对象类型。

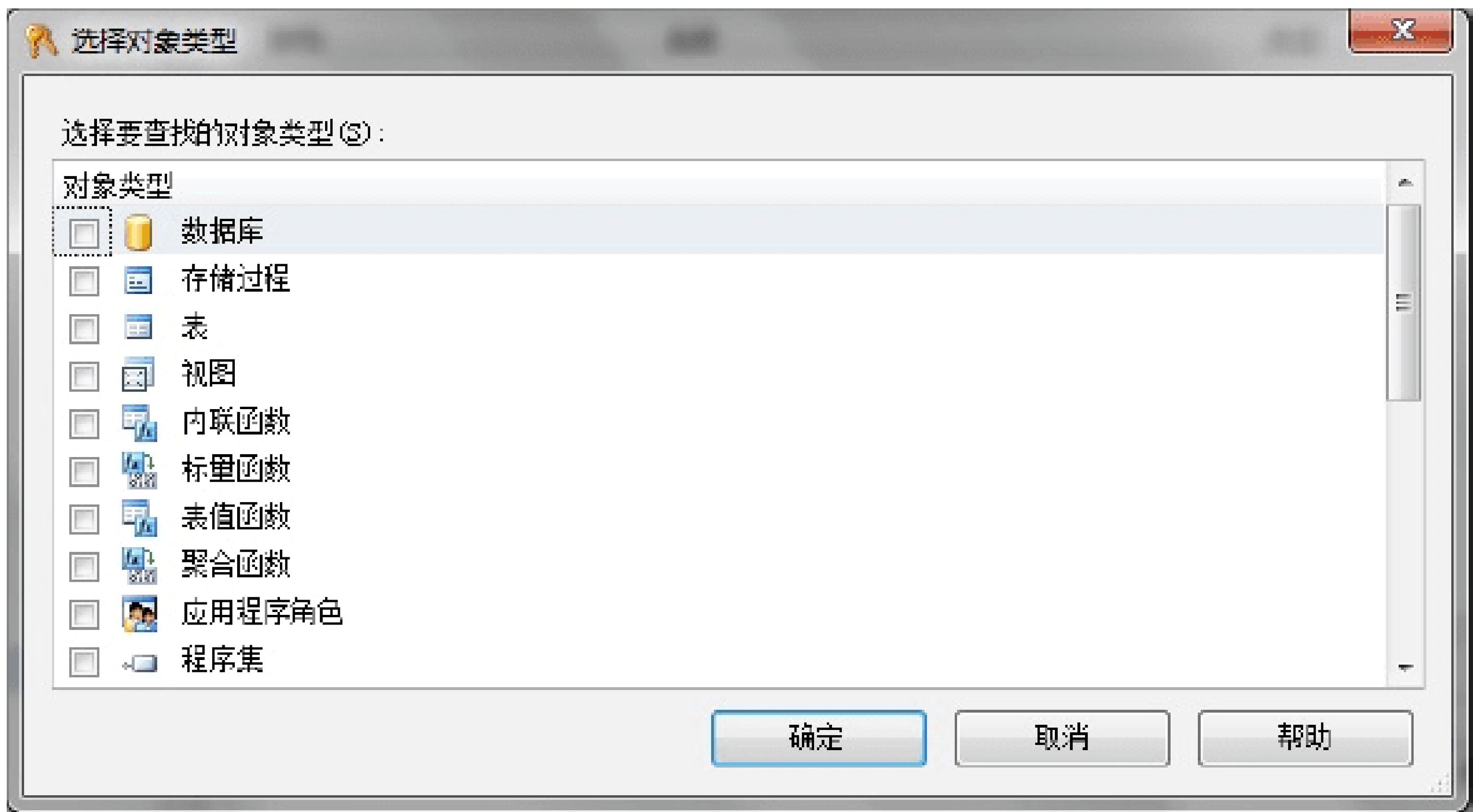


图 18.21　“选择对象类型”对话框

（7）选择“选择对象类型”对话框中的“数据库”选项，单击“确定”按钮返回“数据库用户”窗口，如图 18.22 所示。

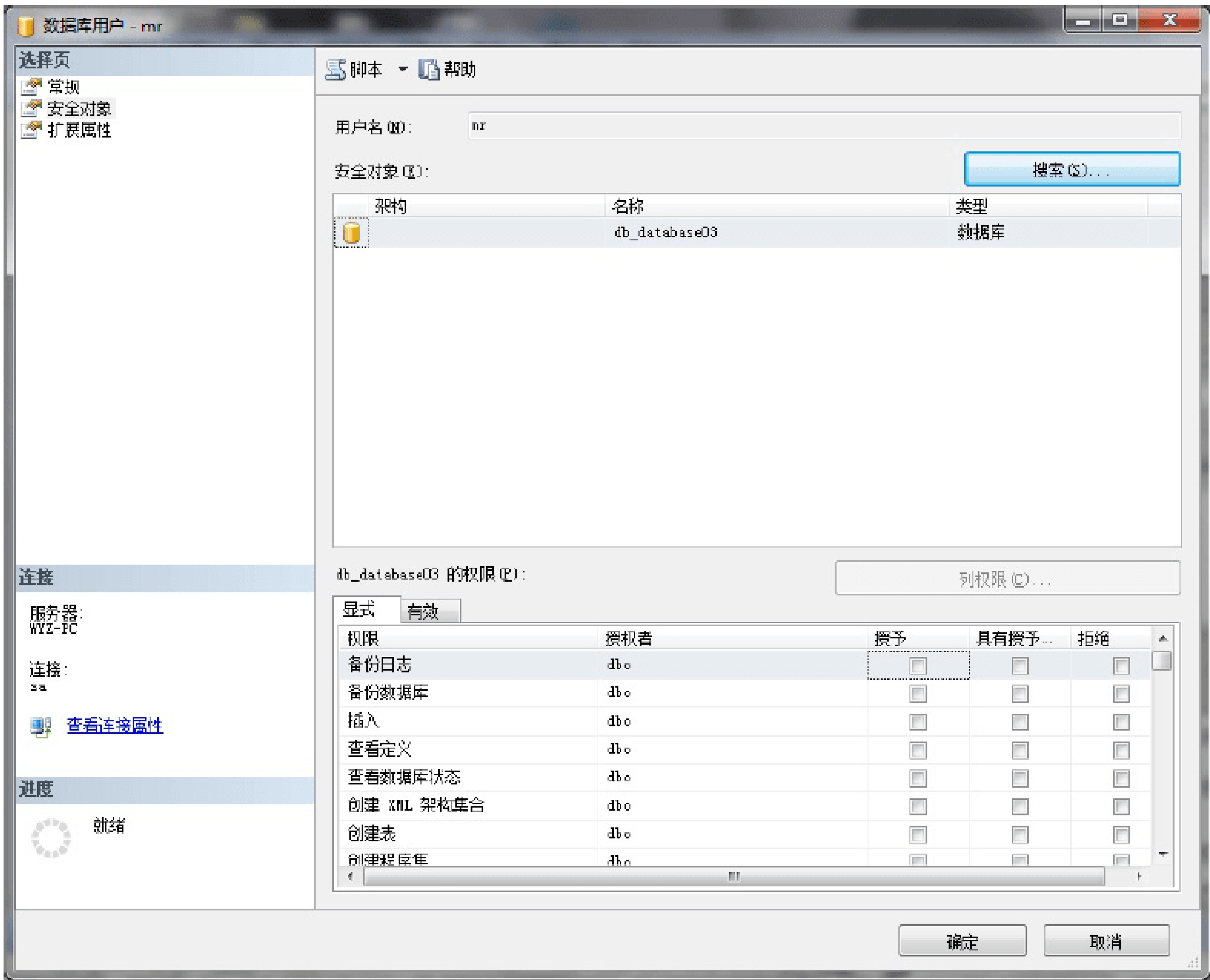


图 18.22　“数据库用户”窗口

（8）在显示权限列表框为该用户选择所需权限，单击“确定”按钮即可将所选权限授予该用户。

2．删除权限

删除权限的操作与授予权限操作基本相同。删除权限的主要步骤如下。

（1）通过“开始”→“程序”→Microsoft SQL Server 2014→SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）在弹出的“连接到数据库引擎”对话框中输入服务器名称，并选择登录服务器使用的身份验

证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开“服务器名称”→“数据库”→“数据库名称”→“安全性”→“用户”，在“用户”上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令。

（4）弹出“数据库用户”窗口，在“选择页”中单击“安全对象”。

（5）单击“添加”按钮，添加访问及操作的对象类型。

（6）在“数据库用户”窗口显示权限列表框取消选中该用户选择所需权限，单击“确定”按钮即可将权限从该用户删除。

18.5 小　　结

本章介绍了加强 SQL Server 2014 安全管理的方式。例如，SQL Server 身份验证、创建数据库用户、SQL Server 角色和 SQL Server 权限。读者应熟悉两种 SQL Server 身份验证模式，并能够创建和管理登录账户，为数据库指定用户，为 SQL Server 角色添加或删除用户；了解授予或删除用户的操作权限的方法。

第19章

SQL Server维护管理

（视频讲解：27分钟）

数据库在使用的过程中，所有的对象（如表、视图和存储过程等）和数据都有可能根据需要随时进行更新，如果数据库出现突发的灾难性事件，导致数据丢失和损坏，后果将不堪设想，所以对数据库的维护工作将是数据库使用过程中一个重要的环节。

学习摘要：

- 数据库的脱机与联机
- 分离和附加数据库
- 导入和导出数据表
- 备份和恢复数据库
- 将数据库或数据表生成脚本
- 执行脚本



19.1　脱机与联机数据库

如果需要暂时关闭某个数据库的服务，用户可以通过选择脱机的方式来实现。脱机后，在需要时可以对暂时关闭的数据库通过联机操作的方式重新启动服务。下面分别介绍如何实现数据库的脱机与联机操作。

19.1.1　脱机数据库

实现数据库脱机的具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要脱机的数据库 MR_KFGL，在弹出的快捷菜单中选择“任务”→“脱机”命令，进入“使数据库脱机”对话框，如图 19.1 和图 19.2 所示。

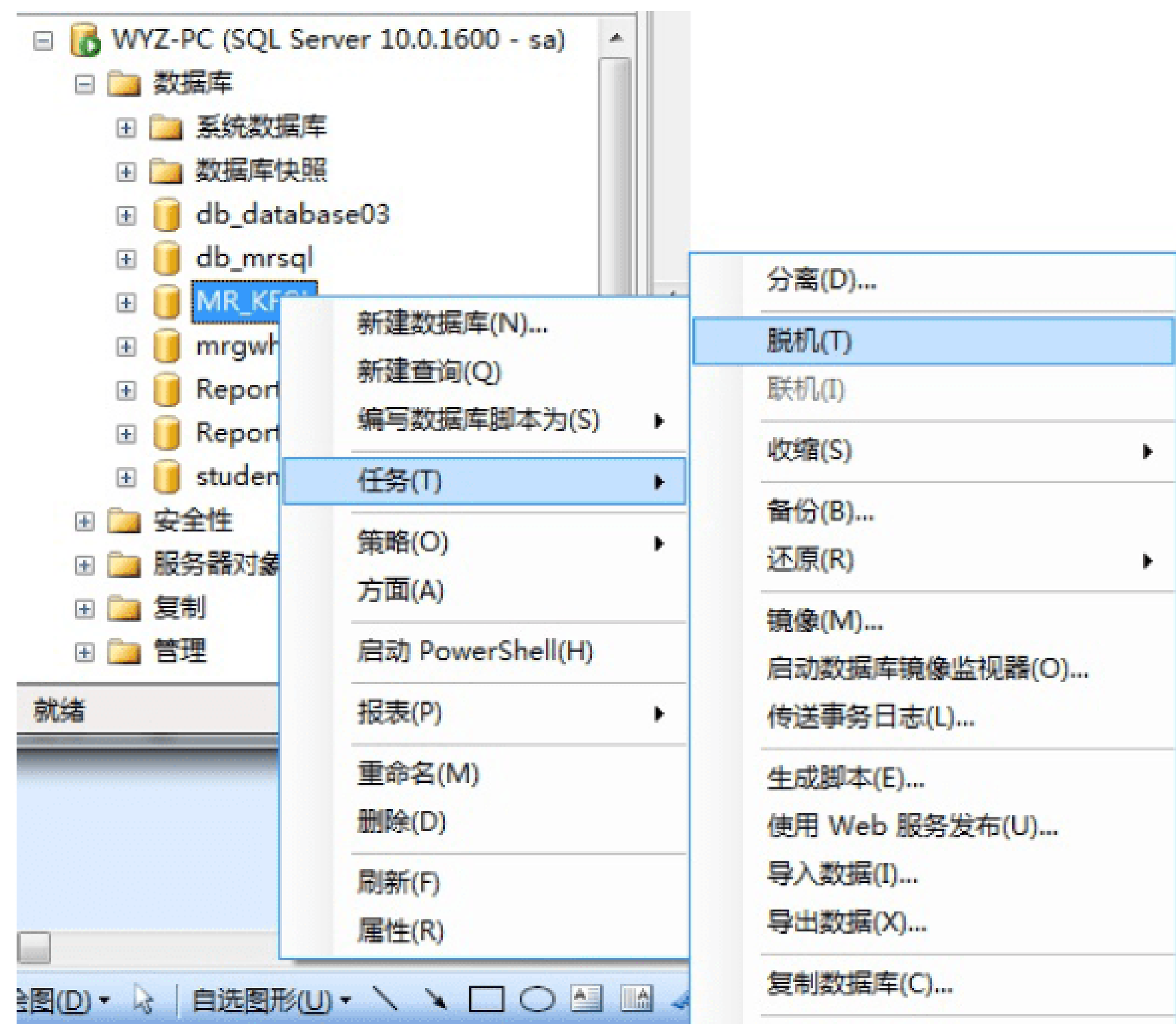


图 19.1　选择脱机数据库



图 19.2　使数据库脱机

（3）脱机完成后，单击“关闭”按钮即可。

19.1.2　联机数据库

实现数据库联机的具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要联机的数据库 MR_KFGL，在弹出的快捷菜单中选择“任务”→“联机”命令，进入“使数据库联机”对话框，如图 19.3 和图 19.4 所示。

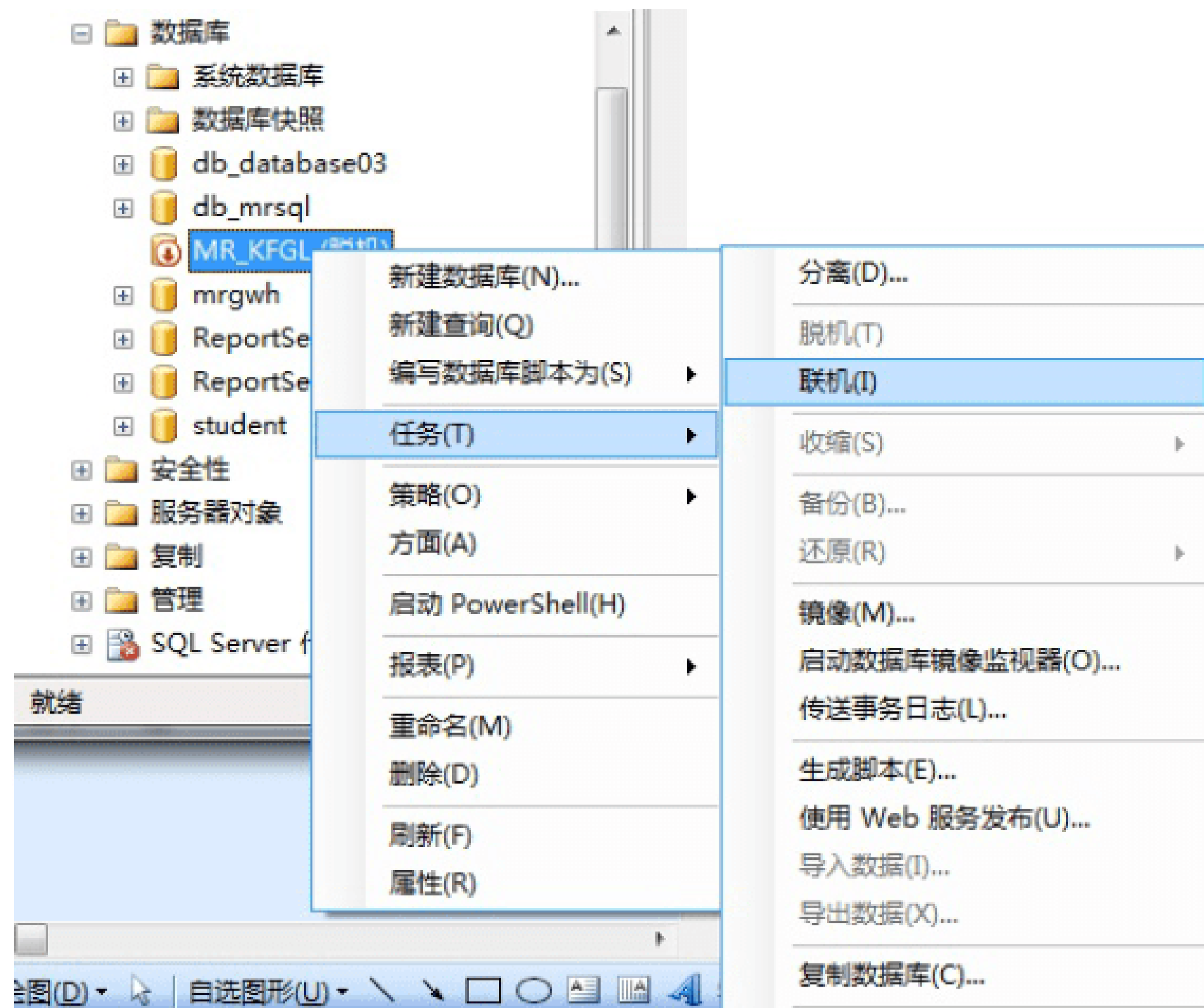


图 19.3　选择联机数据库

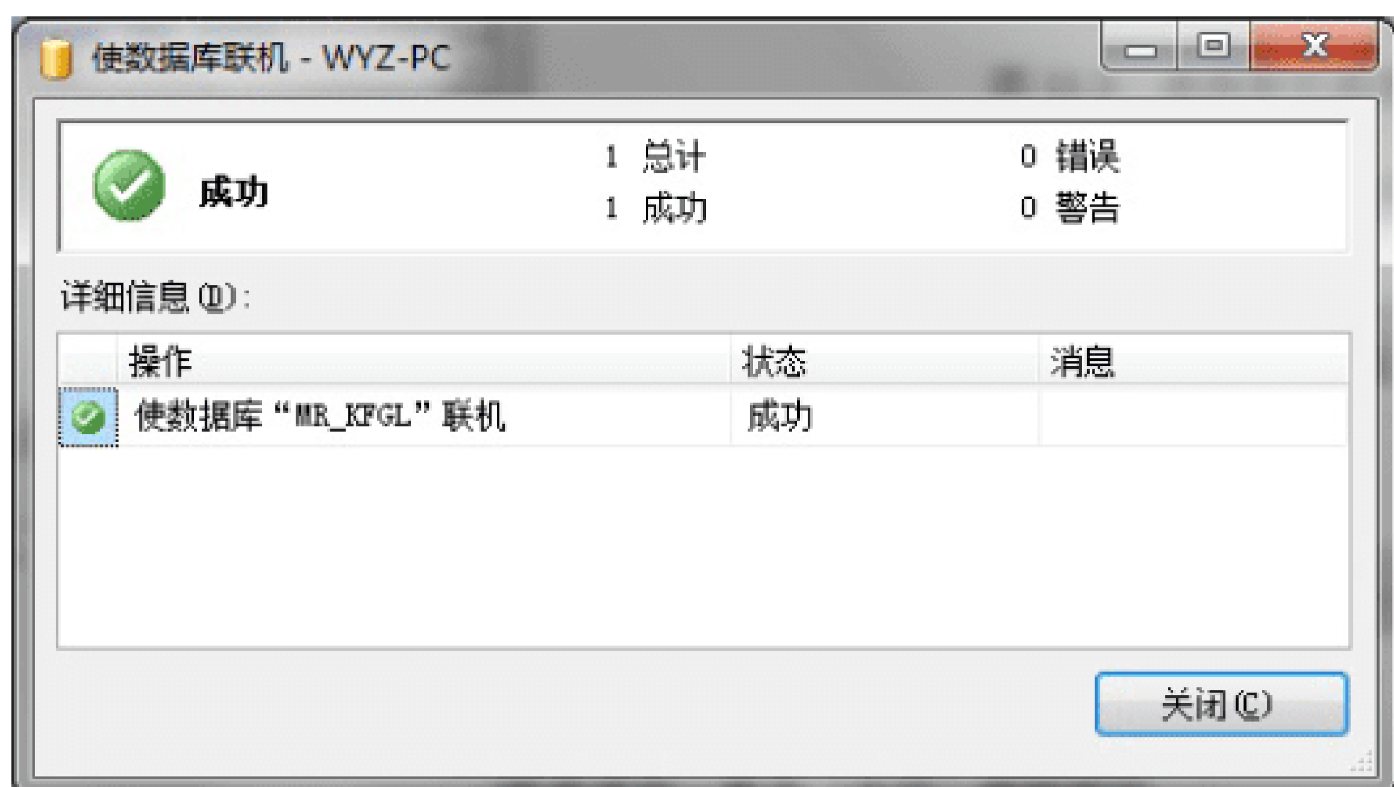


图 19.4　使数据库联机

（3）联机完成后，单击“关闭”按钮即可。

19.2　分离和附加数据库



分离和附加数据库的操作可以将数据库从一台计算机移到另一台计算机，而不必重新创建数据库。除了系统数据库以外，其他数据库都可以从服务器的管理中分离出来，脱离服务器管理的同时保持数据文件和日志文件的完整性和一致性。分离后的数据库又可以根据需要重新附加到数据库服务器中。本节主要介绍如何分离与附加数据库。

19.2.1　分离数据库

分离数据库不是删除数据库，它只是将数据库从服务器中分离出去。下面介绍如何分离数据库 MR_KFGL。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要分离的数据库 MR_KFGL，在弹出的快捷菜单中选择“任务”→“分离”命令，如图 19.5 所示。

（3）进入“分离数据库”窗口，如图 19.6 所示，在“要分离的数据库”列表中选择可以分离的数据库选项。其中，“删除连接”表示是否断开与指定数据库的连接；“更新统计信息”表示在分离数据

库之前是否更新过时的优化统计信息；在此选择“删除连接”“更新统计信息”选项。

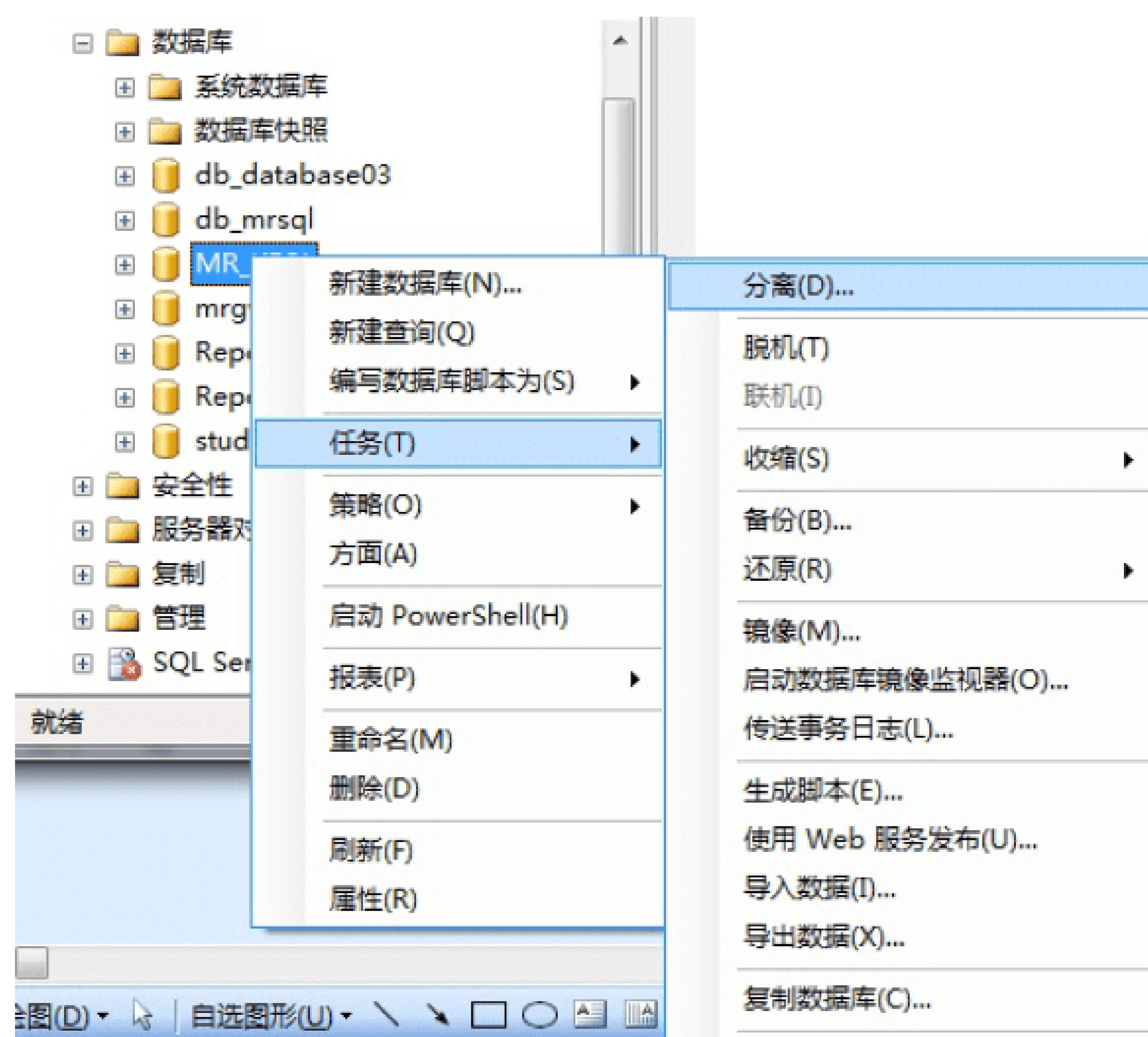


图 19.5　分离数据库

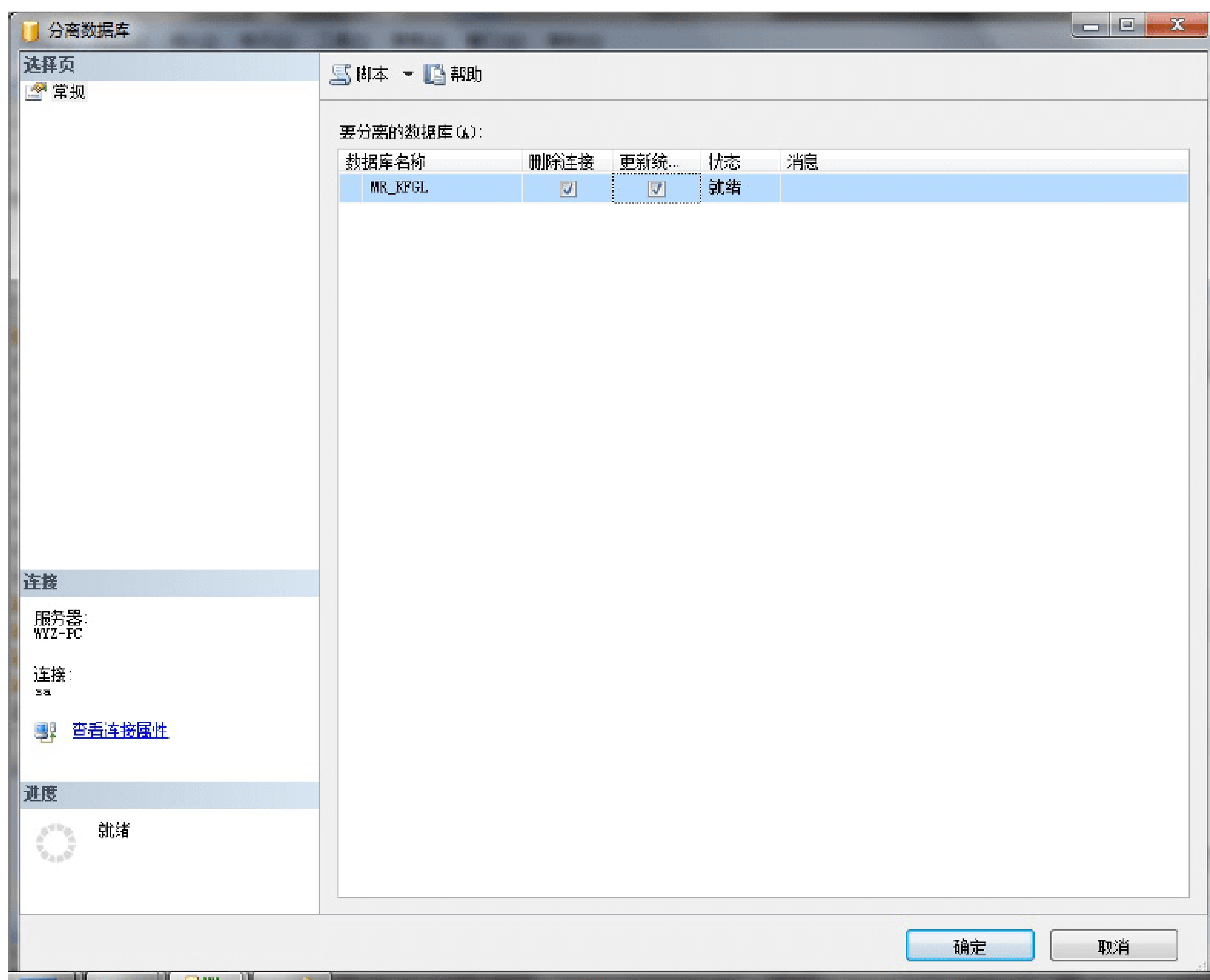


图 19.6　“分离数据库”窗口

（4）单击“确定”按钮完成数据库的分离操作。

19.2.2　附加数据库

与分离操作相对应的就是附加操作，它可以将分离的数据库重新附加到服务器中，也可以附加其他服务器组中分离的数据库。但在附加数据库时必须指定主数据文件（MDF 文件）的名称和物理位置。

下面附加数据库 MR_KFGL，具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击“数据库”选项，在弹出的快捷菜单中选择“附加”命令，如图 19.7 所示。

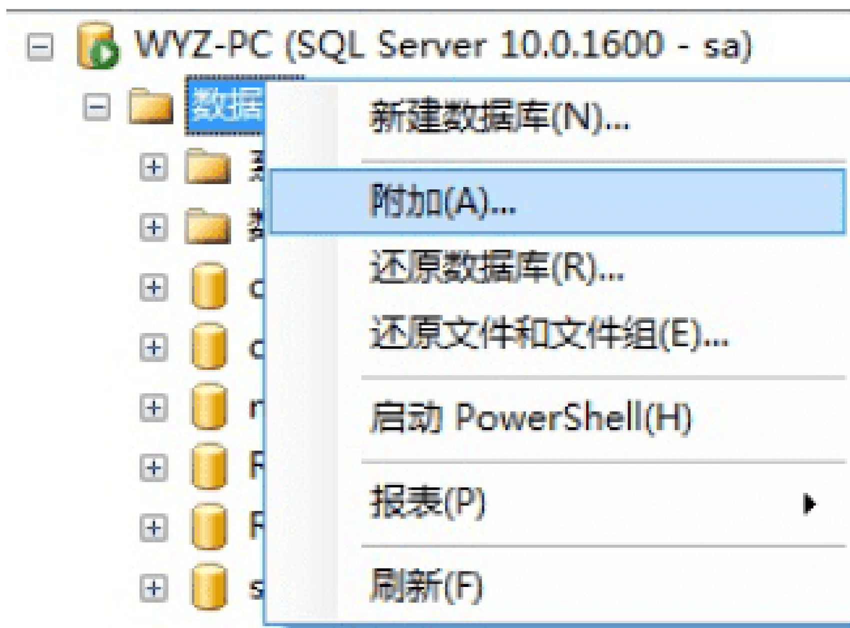


图 19.7　附加数据库

（3）进入“附加数据库”窗口，如图 19.8 所示。单击“添加”按钮，在弹出的“定位数据库文件”对话框中选择要附加的扩展名为.mdf 的数据库文件，单击“确定”按钮后，数据库文件及数据库日志文件将自动添加到列表框中。最后单击“确定”按钮完成数据库附加操作。

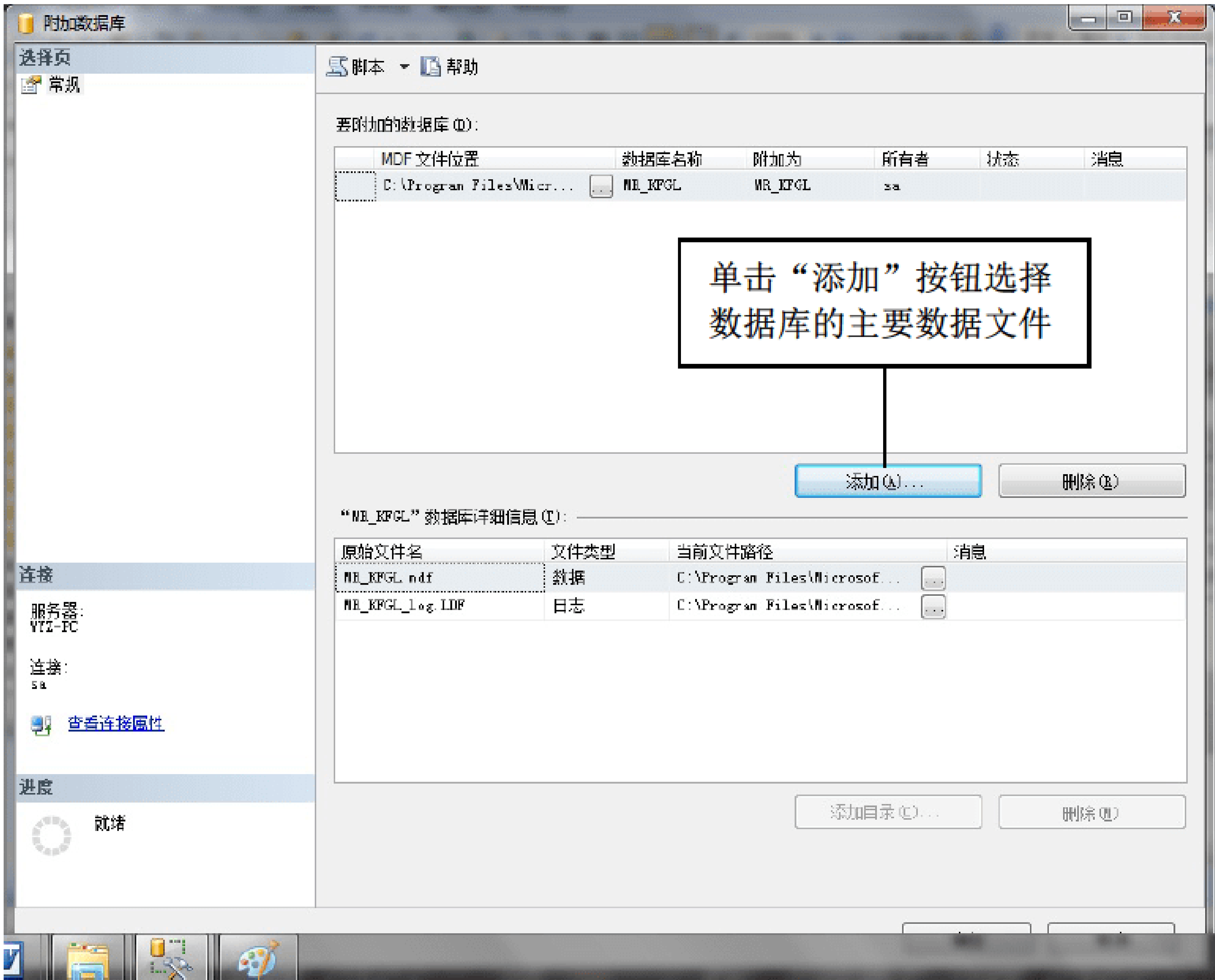


图 19.8　“附加数据库”窗口

19.3　导入和导出数据表



SQL Server 2014 提供了强大的数据导入和导出功能，它可以在多种常用数据格式（数据库、电子

表格和文本文件）之间导入和导出数据，为不同数据源间的数据转换提供了方便。本节主要介绍如何导入和导出数据表。

19.3.1 导入 SQL Server 数据表

导入数据是从 SQL Server 的外部数据源中检索数据，然后将数据插入 SQL Server 表的过程。下面主要介绍通过导入和导出向导将 SQL Server 数据库 student 中的部分数据表导入 SQL Server 数据库 MR_KFGL 中。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击指定的数据库 MR_KFGL 选项，在弹出的快捷菜单中选择“任务”→“导入数据”命令，如图 19.9 所示。

（3）进入“SQL Server 导入和导出向导”窗口，如图 19.10 所示。

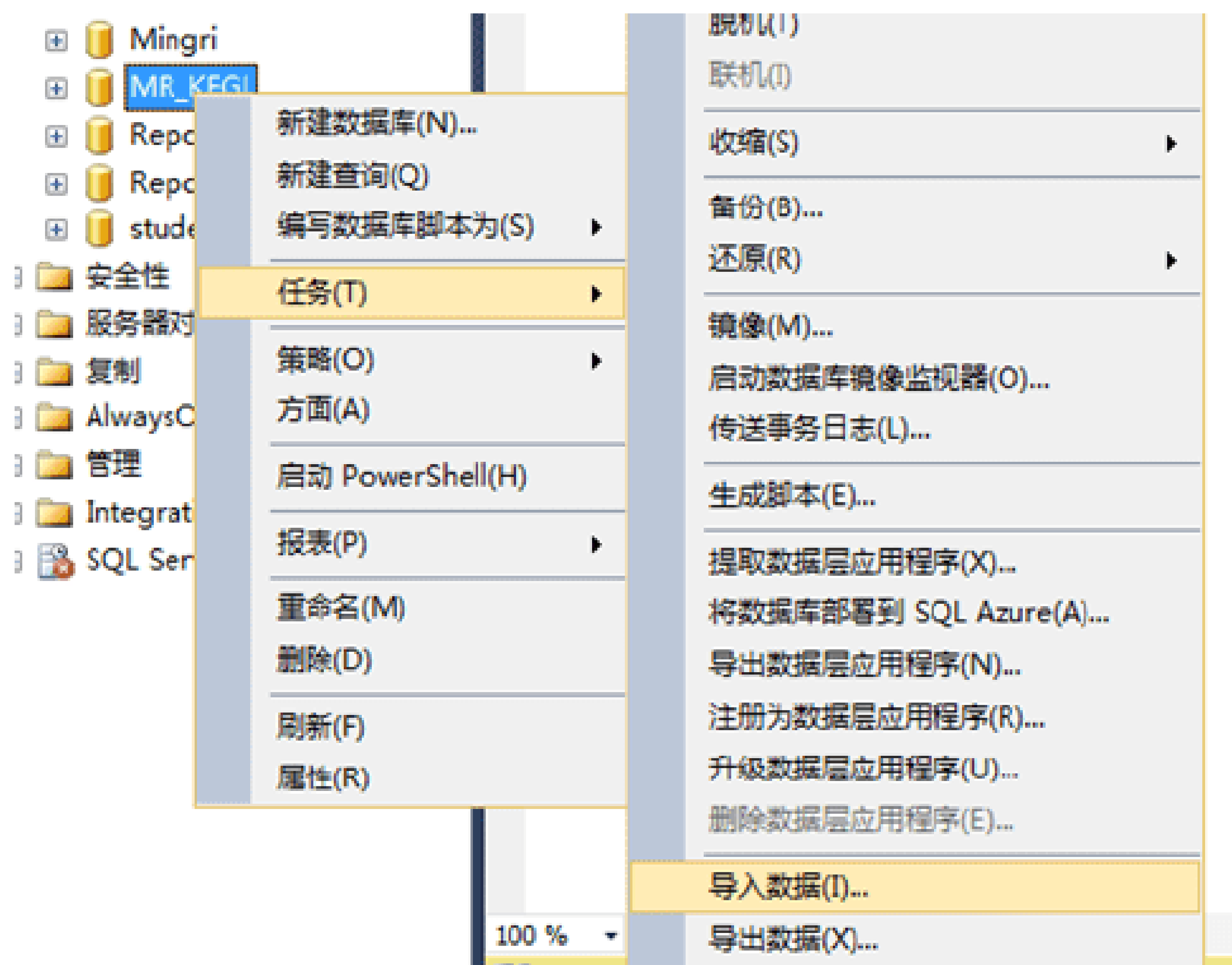


图 19.9　导入数据



图 19.10　SQL Server 导入和导出向导

（4）直接单击“下一步”按钮进入“选择数据源”界面，如图 19.11 所示。首先从“数据源”下拉列表框中选择数据库类型，这里是从 SQL Server 的数据库中导入数据，所以选择默认设置 SQL Server Native Client 10.0 选项即可；然后在“数据库”下拉列表框中选择从哪个数据库导入数据，这里选择数据库 student。

（5）单击“下一步”按钮，进入“选择目标”界面，如图 19.12 所示。这里是将数据导入 SQL Server 数据库，所以在“目标”下拉列表框中选择默认设置 SQL Server Native Client 10.0 选项即可，要导入的目标数据库是 MR_KFGL，所以在“数据库”下拉列表框中选择数据库 MR_KFGL。

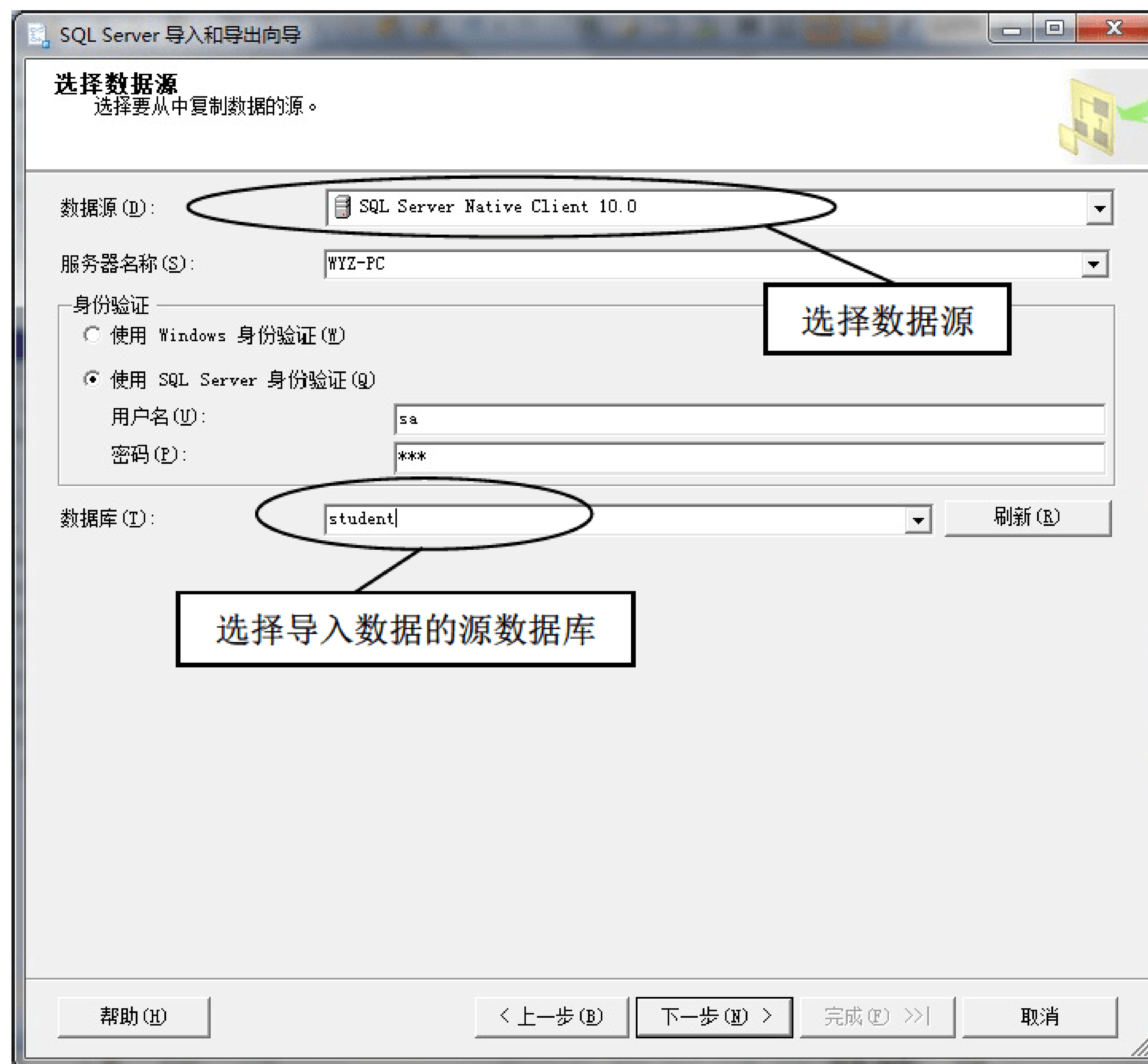


图 19.11　选择数据源



图 19.12　选择目标

（6）单击“下一步”按钮，进入“指定表复制或查询”界面，如图 19.13 所示。



图 19.13　指定表复制或查询

（7）直接单击“下一步”按钮，进入“选择源表和源视图”界面，如图 19.14 所示。这里选择复制 grade 表选项。

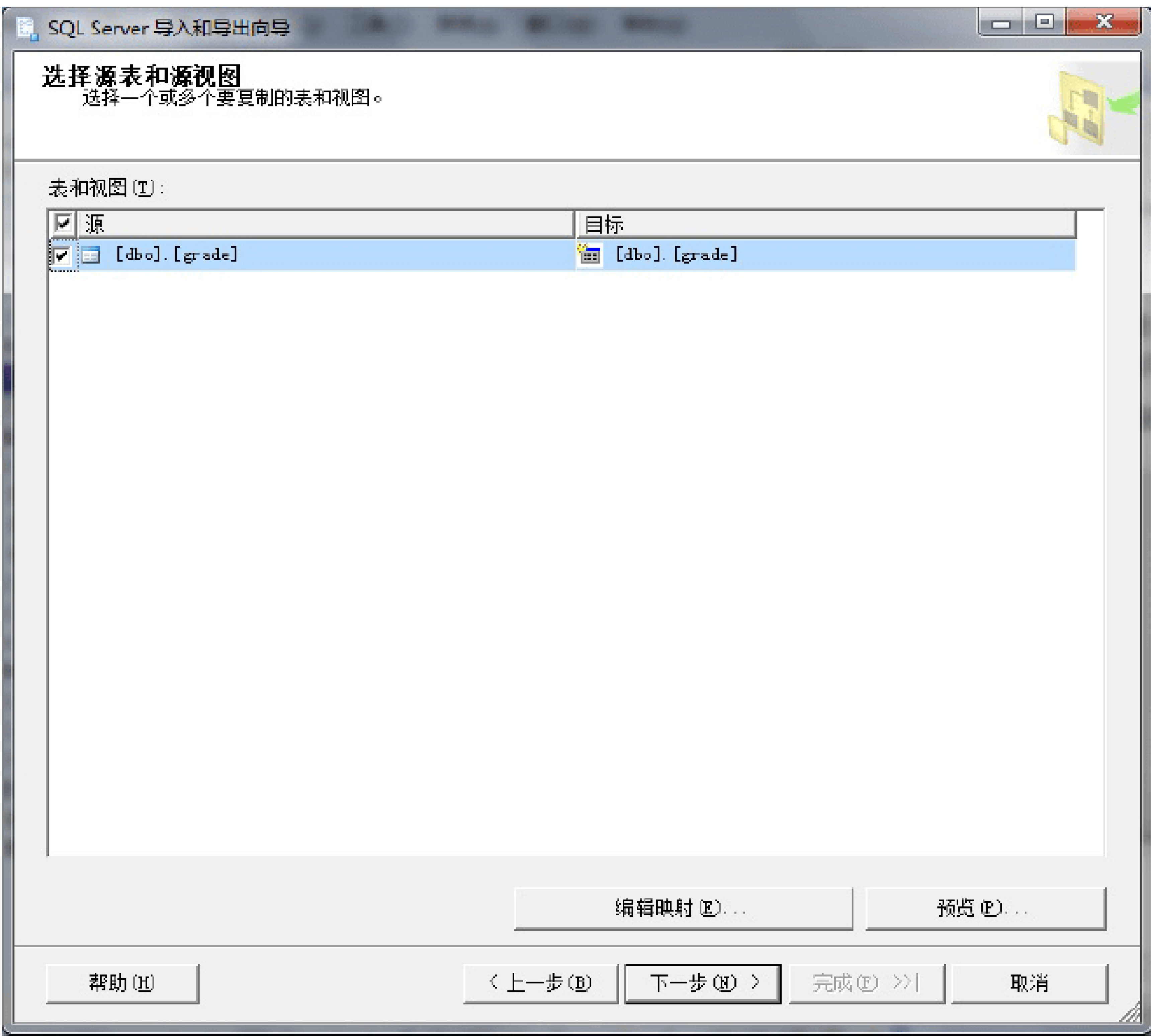


图 19.14　选择源表和源视图

（8）单击“下一步”按钮，进入“保存并运行包”界面，如图 19.15 所示。



图 19.15　保存并运行包

（9）单击“下一步”按钮，进入“完成该向导”界面，如图 19.16 所示。



图 19.16　完成该向导

（10）单击“完成”按钮开始执行复制，如图 19.17 所示。最后单击“关闭”按钮完成数据表的导

入操作。

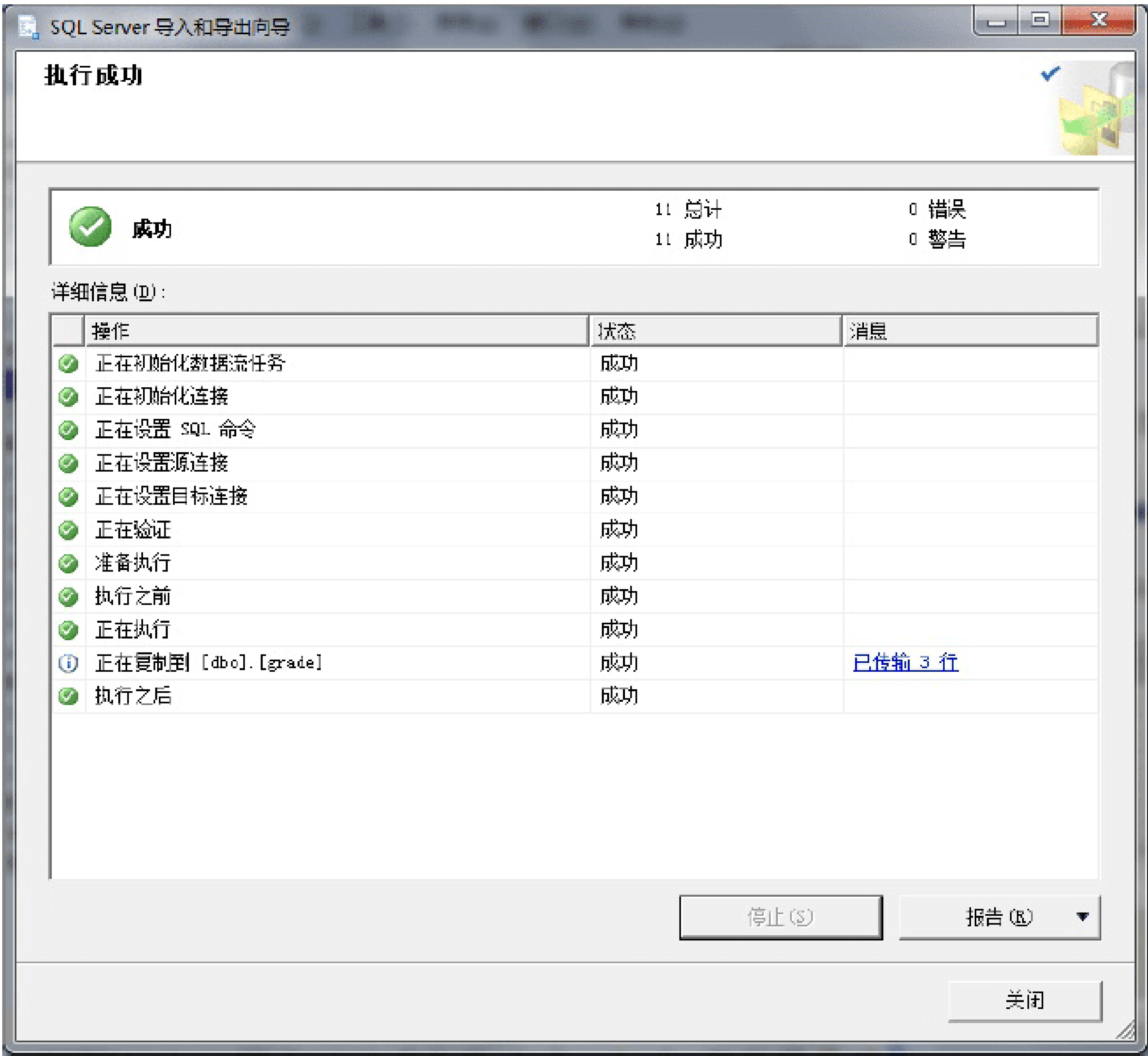


图 19.17 复制成功

（11）展开数据库 MR_KFGL，单击“表”选项，即可查看从数据库 student 中导入的数据表，如图 19.18 所示。

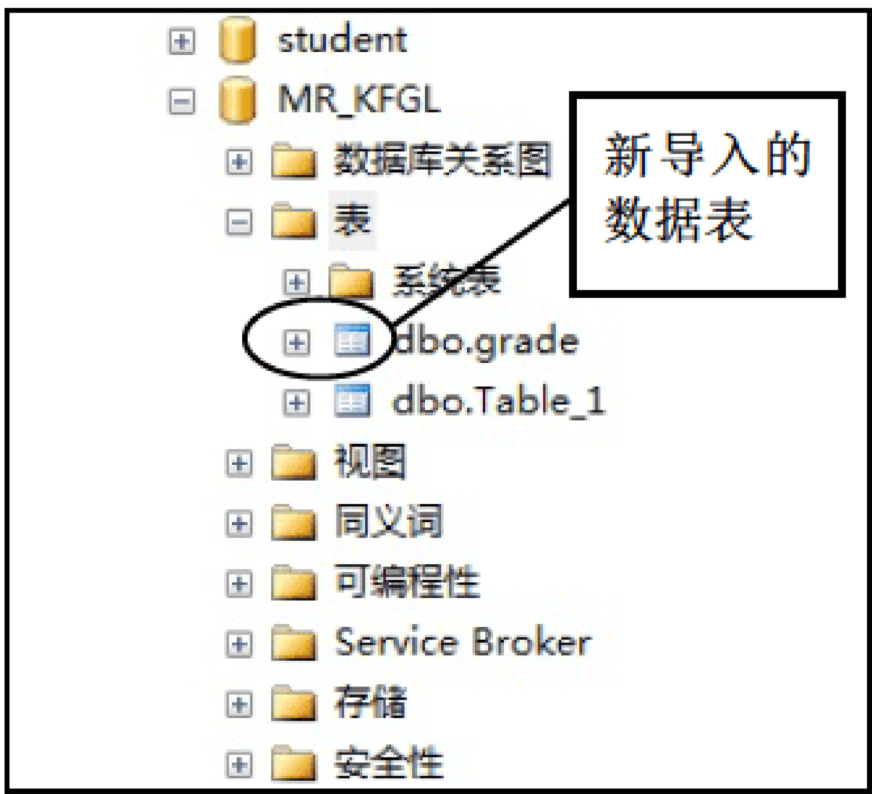


图 19.18 数据表

19.3.2 导出 SQL Server 数据表

导出数据是将 SQL Server 实例中的数据设取为某些用户指定格式的过程，如将 SQL Server 表的内容复制到 Excel 表格中。

下面主要介绍通过导入和导出向导将 SQL Server 数据库 MR_KFGL 中的部分数据表导出到 Excel 表格中。具体操作步骤如下。

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2014 中的数据库。在“对象资源

管理器”中展开“数据库”节点。

（2）鼠标右键单击数据库 MR_KFGL，在弹出的快捷菜单中选择“任务”→“导出数据”命令，如图 19.19 所示，此时将弹出“选择数据源”界面，在该界面中选择要从中复制数据的源，如图 19.20 所示。

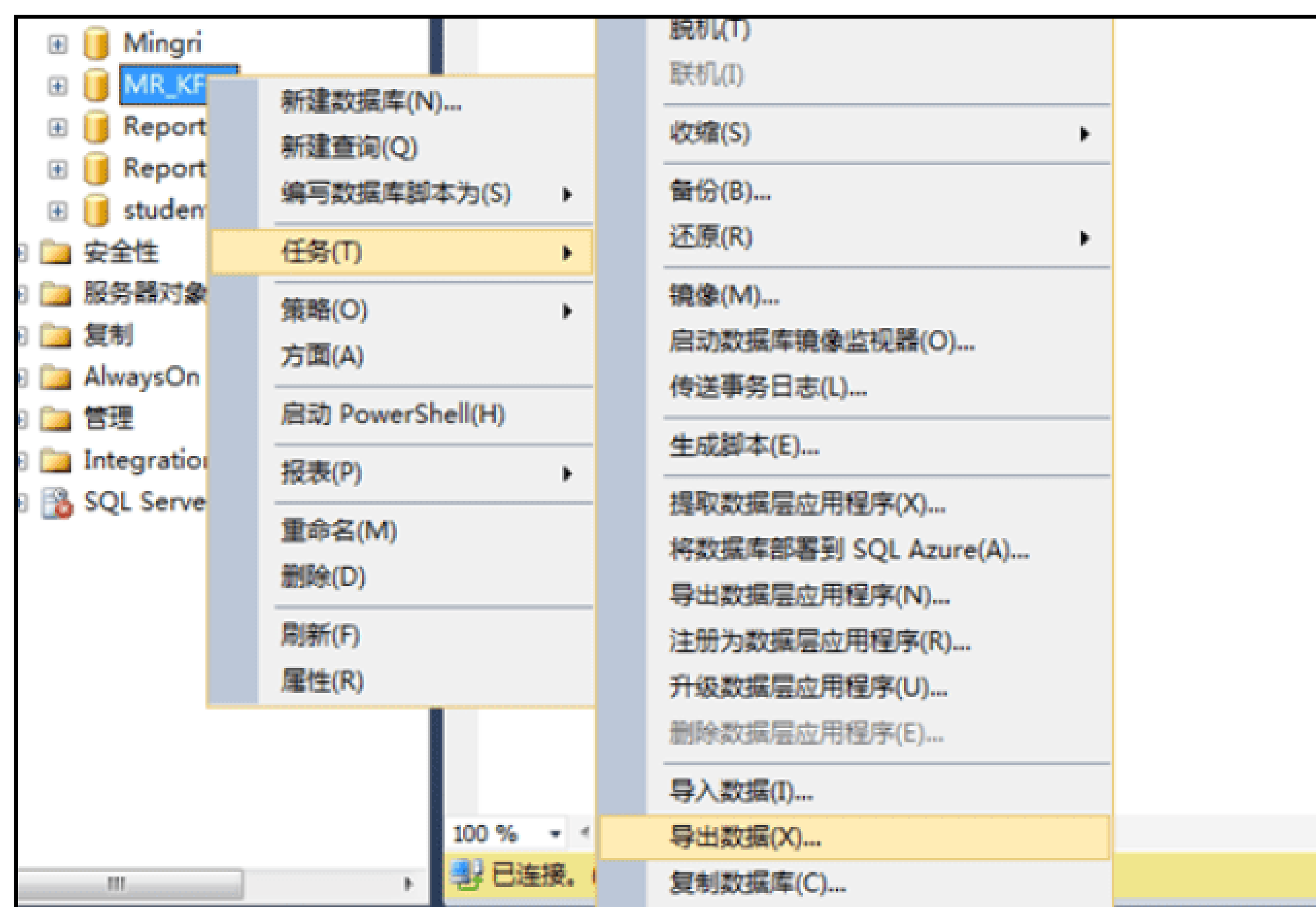


图 19.19　选择“导出数据”命令

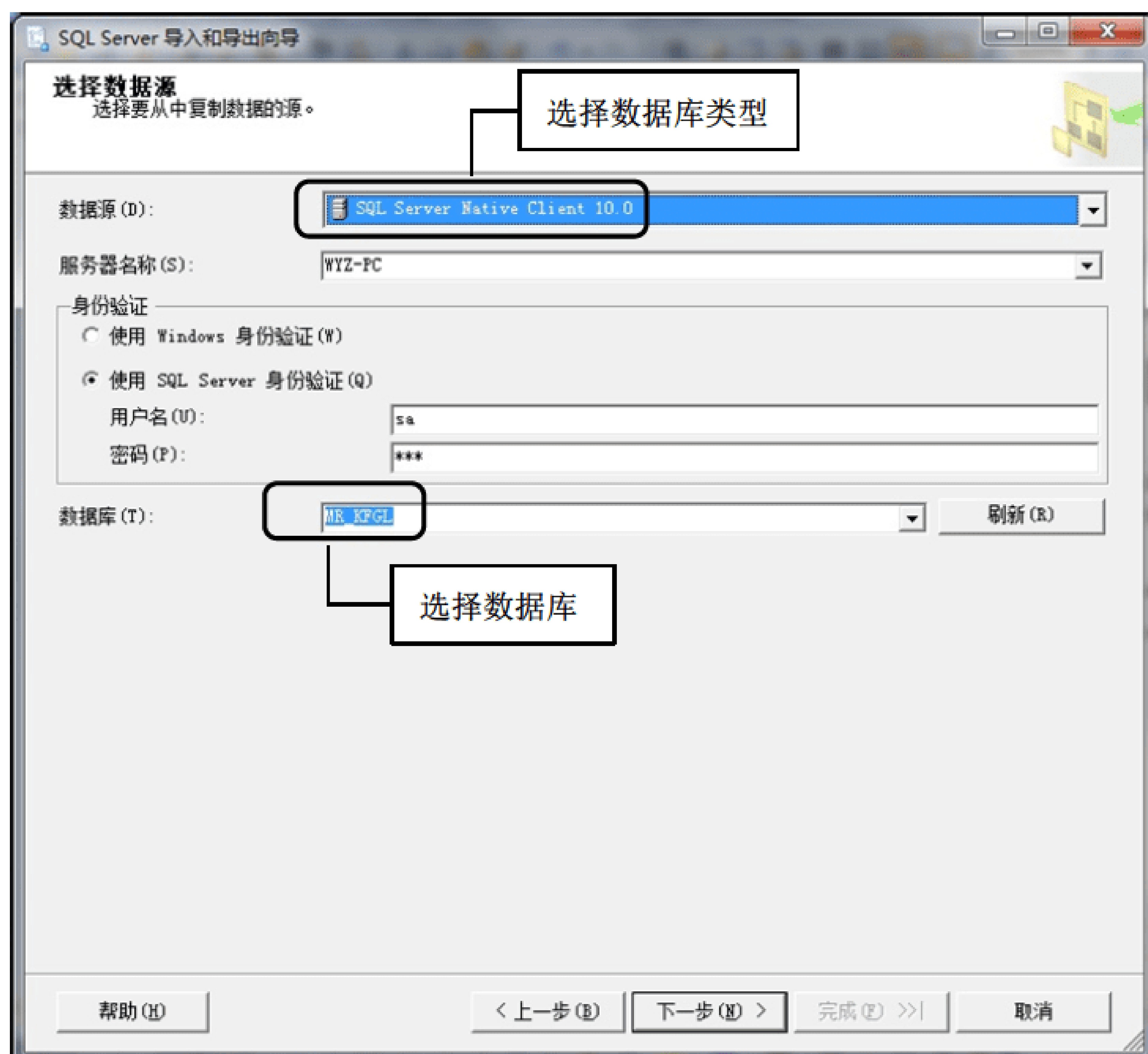


图 19.20　选择数据源

（3）单击“下一步”按钮，进入“选择目标”界面，在该界面中选择要将数据库复制到何处，在该界面中分别选择数据源类型和 Excel 文件的位置，如图 19.21 所示。

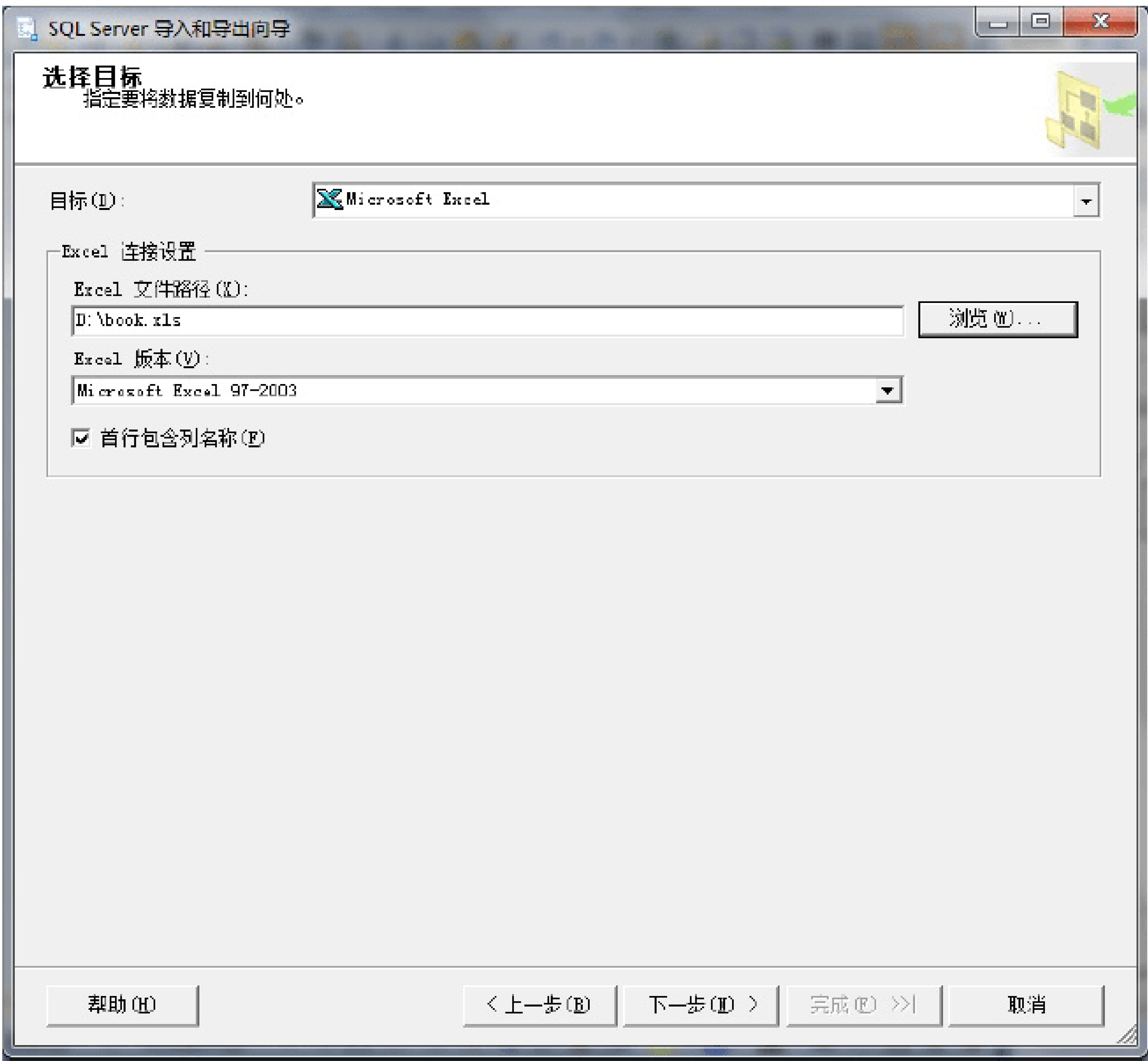


图 19.21　选择目标

（4）单击“下一步”按钮，进入“指定表复制或查询”界面，在该界面中选择是从指定数据源复制一个或多个表和视图，还是从数据源复制查询结果，在这里选中“复制一个或多个表或视图的数据”单选按钮，如图 19.22 所示。



图 19.22　指定表复制或查询

（5）单击“下一步”按钮，进入“选择源表和源视图”界面，在该界面中选择一个或多个要复制的表或视图，这里选择 grade 表，如图 19.23 所示。

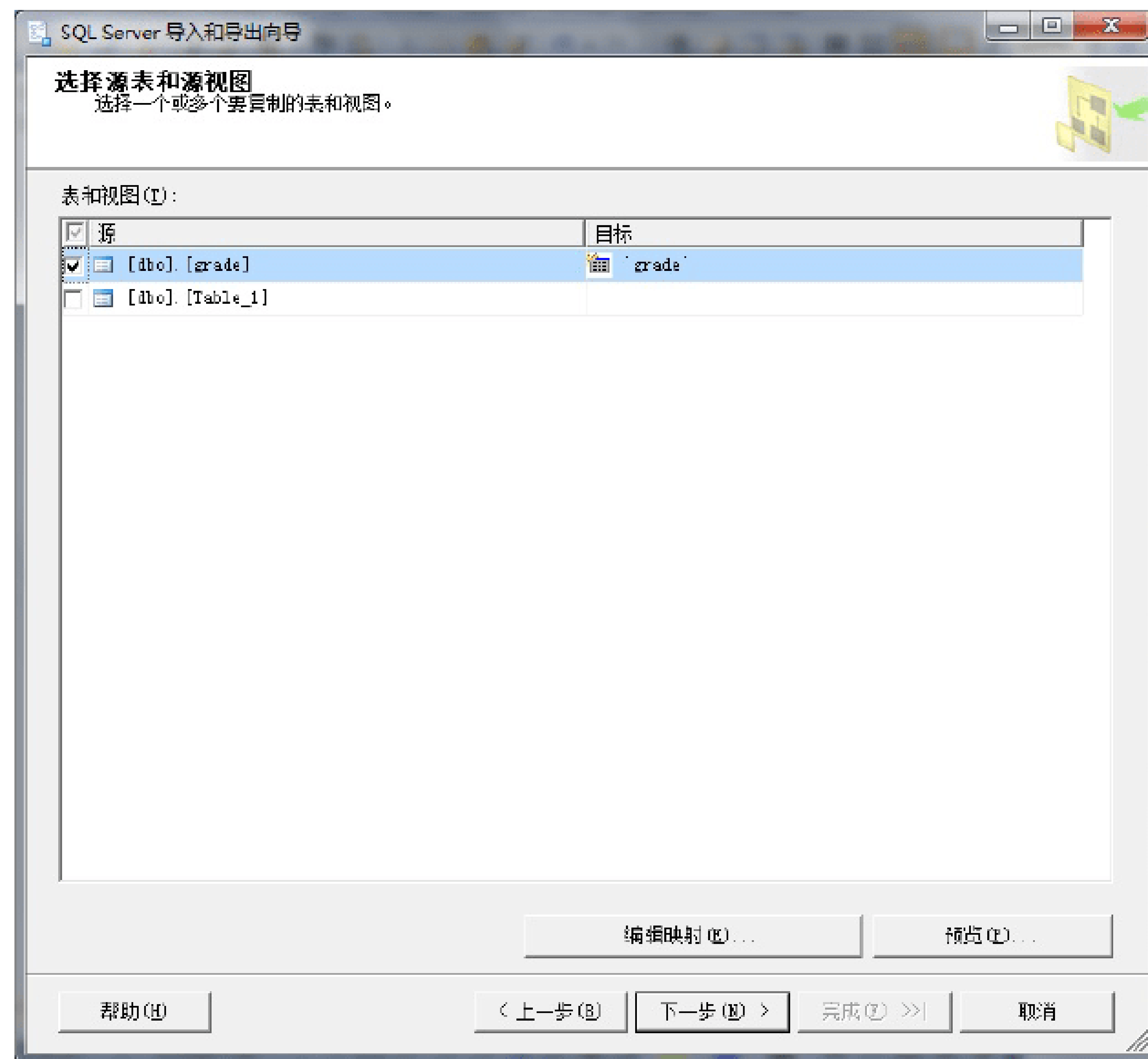


图 19.23　选择源表和源视图

（6）单击“下一步”按钮，进入“保存并运行包”界面，该界面用于提示是否选择 SSIS 包，如图 19.24 所示。



图 19.24　保存并运行包

（7）单击“下一步”按钮，进入“完成该向导”界面，如图 19.25 所示。



图 19.25　完成该向导

（8）单击“完成”按钮开始执行复制操作，进入“执行成功”界面，如图 19.26 所示。

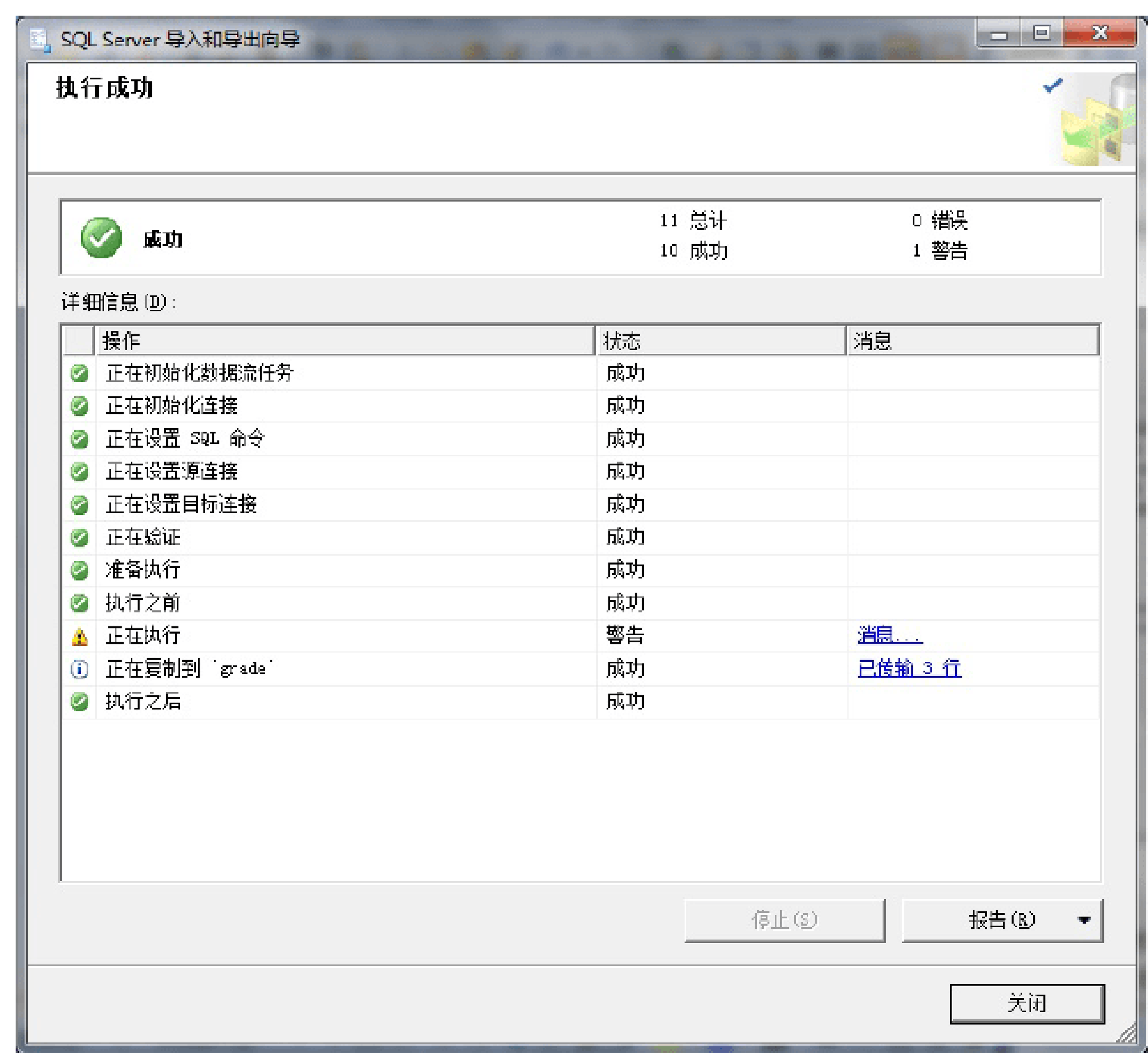


图 19.26　执行成功

（9）最后单击“关闭”按钮，完成数据表的导入操作。

（10）打开 book.xls，即可查看从数据库 MR_KFGL 中导入的数据表中的内容，如图 19.27 所示，如图 19.28 所示为 grade 表中的内容。

	A	B	C	D
1	学号	课程代号	课程成绩	学期
2	B005	K02	93.2	2
3	B003	K03	98.3	1
4	B001	K01	96.7	1
5				

图 19.27　Excel 文件中的内容

学号	课程代号	课程成绩	学期
B005	K02	93.2	2
B003	K03	98.3	1
B001	K01	96.7	1
NULL	NULL	NULL	NULL

图 19.28　grade 表中的内容

19.4　备份和恢复数据库


视频讲解

对于数据库管理员来说，备份和恢复数据库是保证数据库安全性的一项重要工作。SQL Server 2014 提供了高性能的备份和恢复功能，它可以实现多种方式的数据库备份和恢复操作，避免了由于各种故障造成的数据损坏或丢失。本节主要介绍如何实现数据库的备份与恢复操作。

19.4.1　备份类型

“备份”是数据的副本，用于在系统发生故障后还原和恢复数据。SQL Server 2014 提供了 3 种常用的备份类型：数据库备份、差异数据库备份和事务日志备份，下面分别对其进行介绍。

1．数据库备份

数据库备份包括完整备份和完整差异备份。它简单、易用，适用于所有数据库，与事务日志备份和差异数据库备份相比，数据库备份中的每个备份使用的存储空间更多。

（1）完整备份：完整备份包含数据库中的所有数据，可以用作完整差异备份所基于的“基准备份”。

（2）完整差异备份：完整差异备份仅记录自前一完整备份后发生更改的数据。

相比之下，完整差异备份速度快，便于进行频繁备份，降低丢失数据的风险。

2．差异数据库备份

差异数据库备份只记录自上次数据库备份后发生更改的数据。其比数据库备份小，并且备份速度快，可以进行经常的备份。

在下列情况中，建议使用差异数据库备份。

（1）自上次数据库备份后，数据库中只有相对较少的数据发生了更改。

（2）使用的是简单恢复模型，希望进行更频繁的备份，但不希望进行频繁的完整数据库备份。

（3）使用的是完全恢复模型或大容量日志记录恢复模型，希望在还原数据库时前滚事务日志备份的时间最少。

3．事务日志备份

事务日志是自上次备份事务日志后对数据库执行的所有事务的一系列记录。使用事务日志备份可以将数据库恢复到故障点或特定的即时点。一般情况下，事务日志备份比数据库备份使用的资源少。

可以经常地创建事务日志备份，以减小丢失数据的危险。

若要使用事务日志备份，必须满足下列要求。

（1）必须先还原前一个完整备份或完整差异备份。

（2）必须按时间顺序还原完整备份或完整差异备份之后创建的所有事务日志。如果此事务日志链中的事务日志备份丢失或损坏，则用户只能还原丢失的事务日志之前的事务日志。

（3）数据库尚未恢复。直到应用完最后一个事务日志之后，才能恢复数据库。如果在还原其中一个中间事务日志备份（日志链结束之前的备份）后恢复数据库，则除非从完整备份开始重新启动整个还原顺序，否则不能还原该备份点之后的数据库。建议用户在恢复数据库之前还原所有的事务日志，然后再另行恢复数据库。

19.4.2 恢复类型

SQL Server 提供了 3 种恢复类型，用户可以根据数据库的可用性和恢复要求选择适合的恢复类型。

（1）简单恢复：允许将数据库恢复到最新的备份。

简单恢复仅用于测试和开发数据库或包含的大部分数据为只读的数据库。简单恢复所需的管理最少，数据只能恢复到最近的完整备份或差异备份，不备份事务日志，且使用的事务日志空间最小。

与以下两种恢复类型相比，简单恢复更容易管理，但如果数据文件损坏，出现数据丢失的风险系数会更高。

（2）完全恢复：允许将数据库恢复到故障点状态。

完全恢复提供了最大的灵活性，使数据库可以恢复到早期时间点，在最大范围内防止出现故障时丢失数据。与简单恢复类型相比，完全恢复模式和大容量日志恢复模式会向数据提供更多的保护。

（3）大容量日志记录恢复：允许大容量日志记录操作。

大容量日志恢复模式是对完全恢复模式的补充。对某些大规模操作（如创建索引或大容量复制），它比完全恢复模式性能更高，占用的日志空间会更少。不过，大容量日志恢复模式会降低时点恢复的灵活性。

19.4.3 备份数据库

“备份数据库”任务可执行不同类型的 SQL Server 数据库备份（完整备份、差异备份和事务日志备份）。

下面以备份数据库 Mingri 为例介绍如何备份数据库。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要备份的数据库 Mingri 选项，在弹出的快捷菜单中选择“任务”→“备份”命令，如图 19.29 所示，进入“备份数据库”窗口，如图 19.30 所示。

（3）可以单击“确定”按钮，直接完成备份（本书是直接单击“确定”按钮完成备份的）。也可以在“目标”面板中更改备份文件的保存位置。单击“添加”按钮，弹出“选择备份目标”对话框，如图 19.31 所示，这里选中“文件名”单选按钮，单击其后的…按钮，设置文件名及其路径。

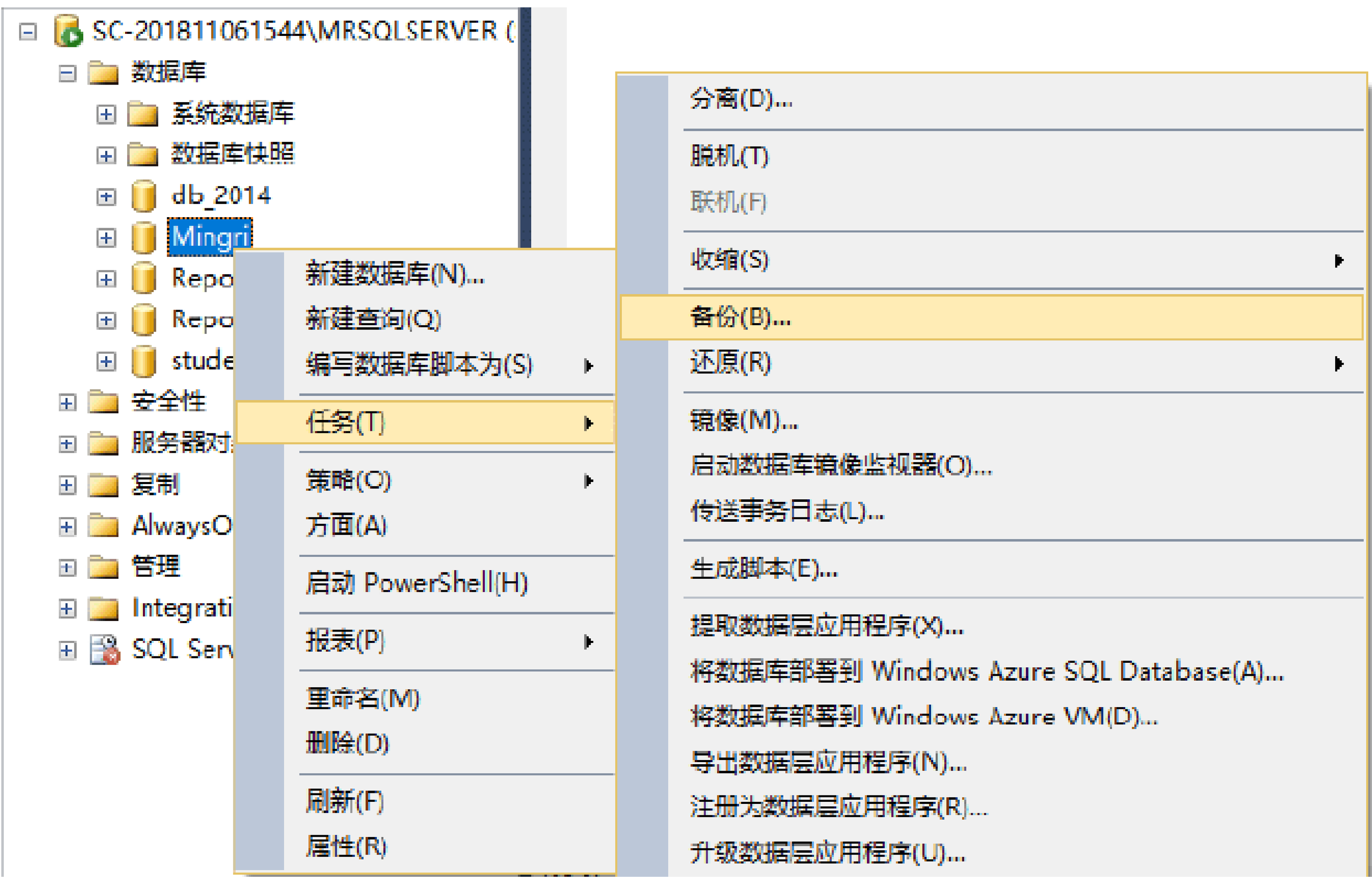


图 19.29　备份数据库

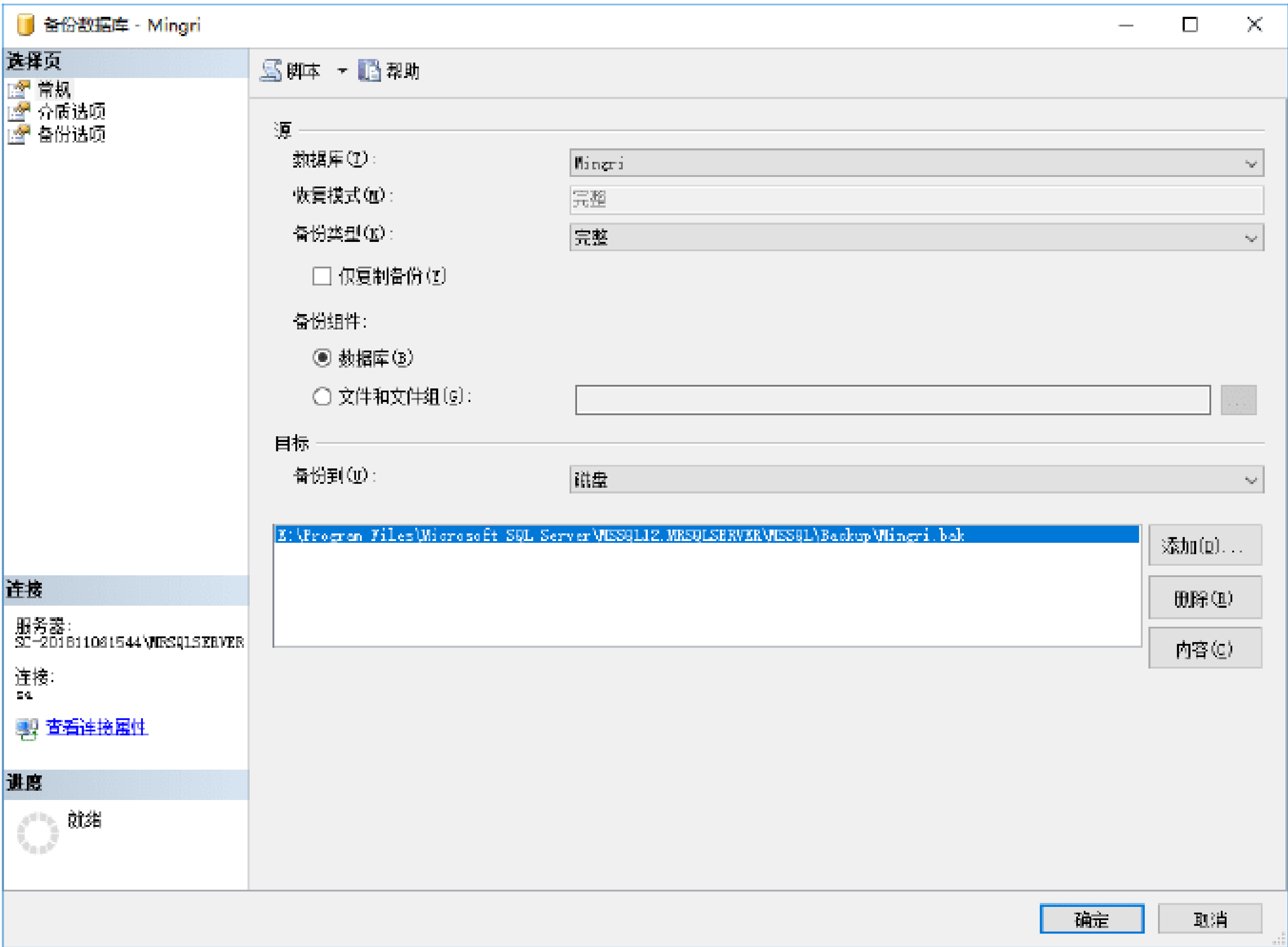


图 19.30　备份数据库

（4）单击“确定”按钮，系统提示备份成功的提示信息，如图 19.32 所示。单击“确定”按钮后即可完成数据库的完整备份。

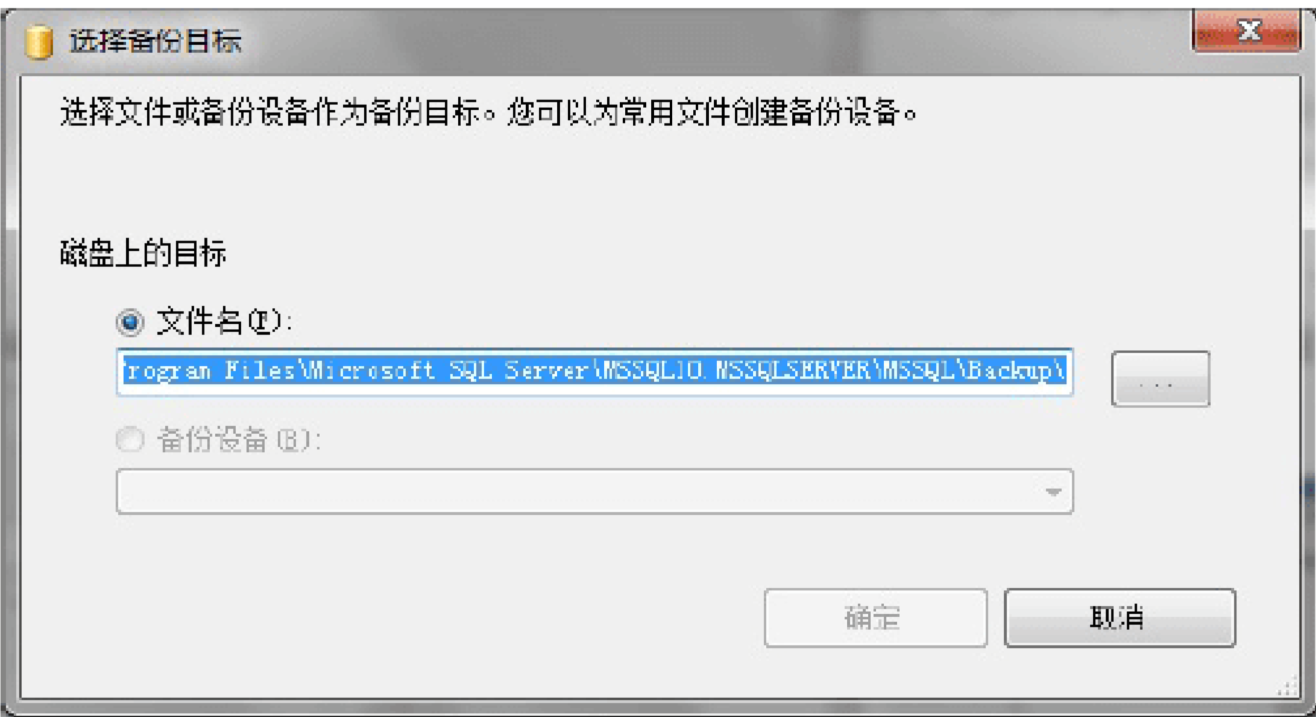


图 19.31　选择备份目标

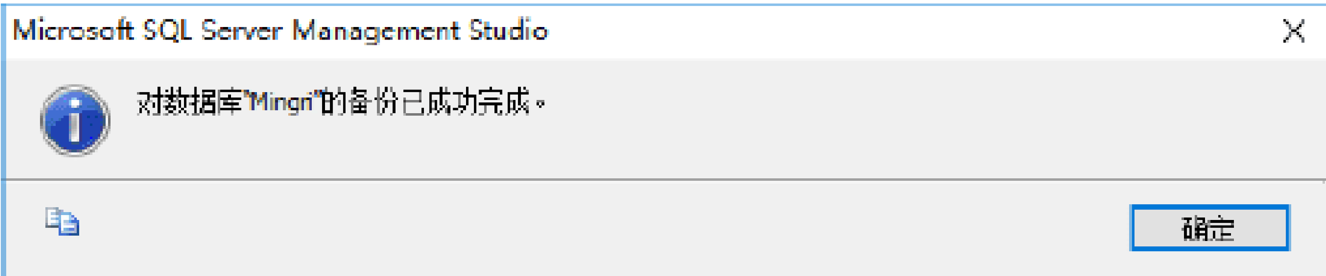


图 19.32　提示信息

19.4.4 恢复数据库

执行数据库备份的目的是便于进行数据恢复。如果发生机器故障、用户误操作等，用户就可以对备份过的数据库进行恢复。

下面介绍如何恢复数据库 Mingri。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要恢复的数据库 Mingri，在弹出的快捷菜单中选择“任务”→“还原”→“数据库”命令，如图 19.33 所示。



图 19.33　恢复数据库

（3）进入“还原数据库”窗口，如图 19.34 所示。在“常规”选项卡中设置还原数据库的名称及源数据库。

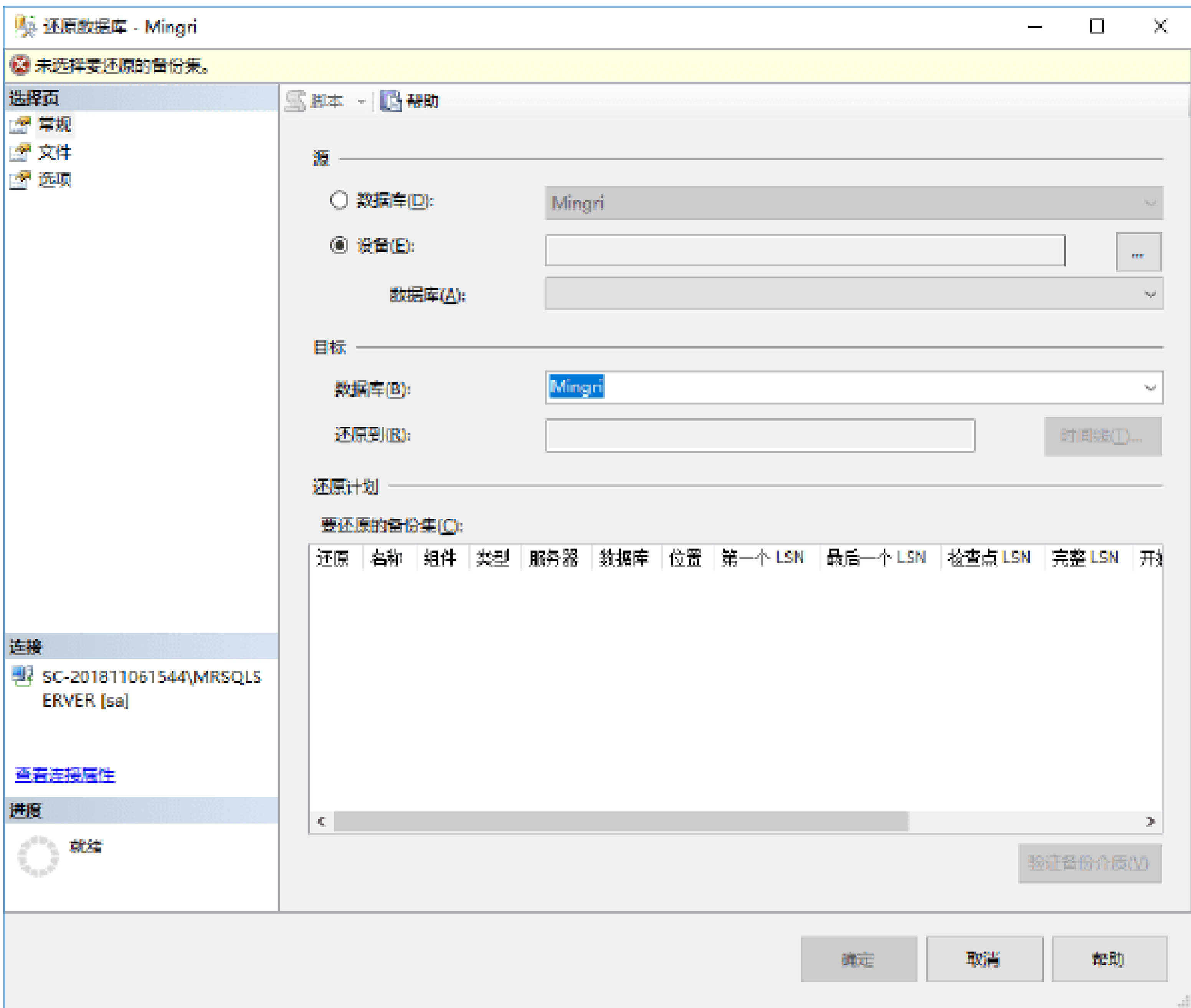


图 19.34　还原数据库

（4）在“源”选项组中选中“设备”单选按钮，然后单击后面的 ... 按钮，这时弹出“选择备份设备”窗口，如图 19.35 所示。

（5）在“备份介质类型”下拉列表框中选择“文件”，单击“添加”按钮，在弹出的“定位备份文件”窗口中选择要恢复的数据库备份文件，然后单击“确定”按钮，如图 19.36 所示。

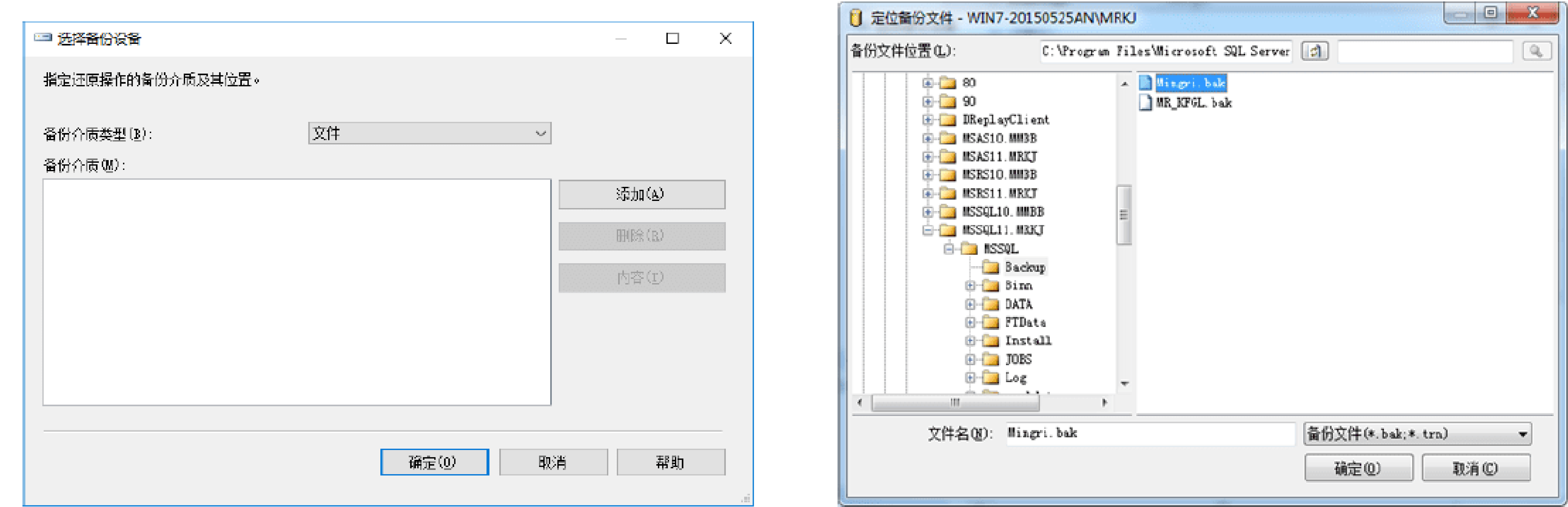


图 19.35　选择备份设备　　　　图 19.36　定位备份文件

（6）回到“选择备份设备”窗口，如图 19.37 所示，单击“确定”按钮，在“还原数据库”窗口中单击“确定”按钮，如图 19.38 所示。最后数据库还原成功，如图 19.39 所示。

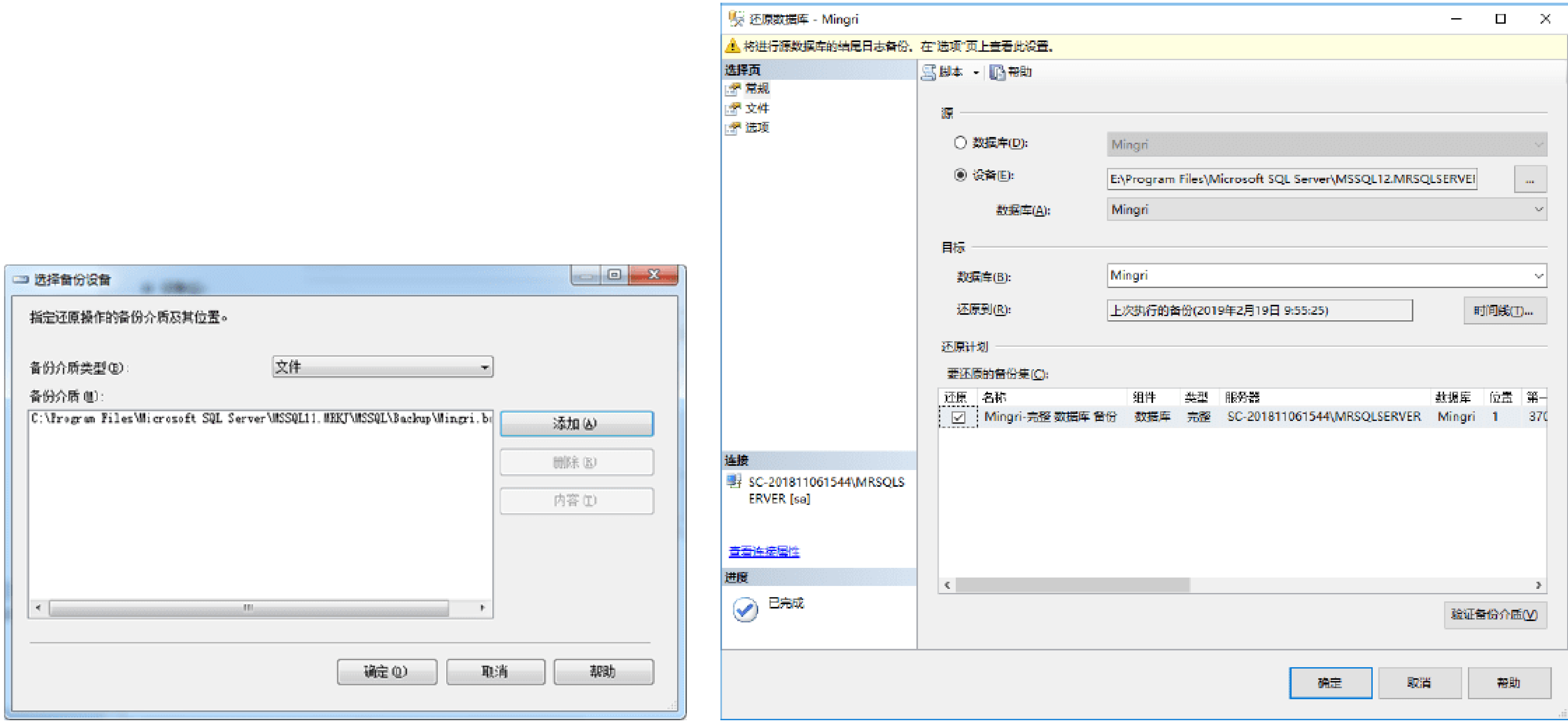


图 19.37　选择备份设备　　　　图 19.38　还原数据库

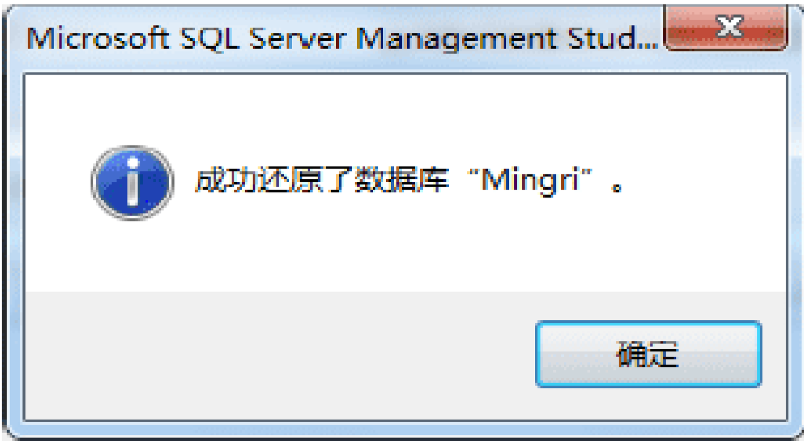


图 19.39　完成还原数据库



19.5 脚　　本

脚本是存储在文件中的一系列 SQL 语句，是可再用的模块化代码。用户通过 SQL Server Management Studio 工具可以对指定文件中的脚本进行修改、分析和执行。

本节主要介绍如何将数据库、数据表生成脚本，以及如何执行脚本。

19.5.1 将数据库生成脚本

数据库在生成脚本文件后，可以在不同的计算机之间传送。下面将数据库 MR_KFGL 生成脚本文件。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击指定的数据库 MR_KFGL，在弹出的快捷菜单中选择“编写数据库脚本为”→“CREATE 到”→“文件”命令，如图 19.40 所示。

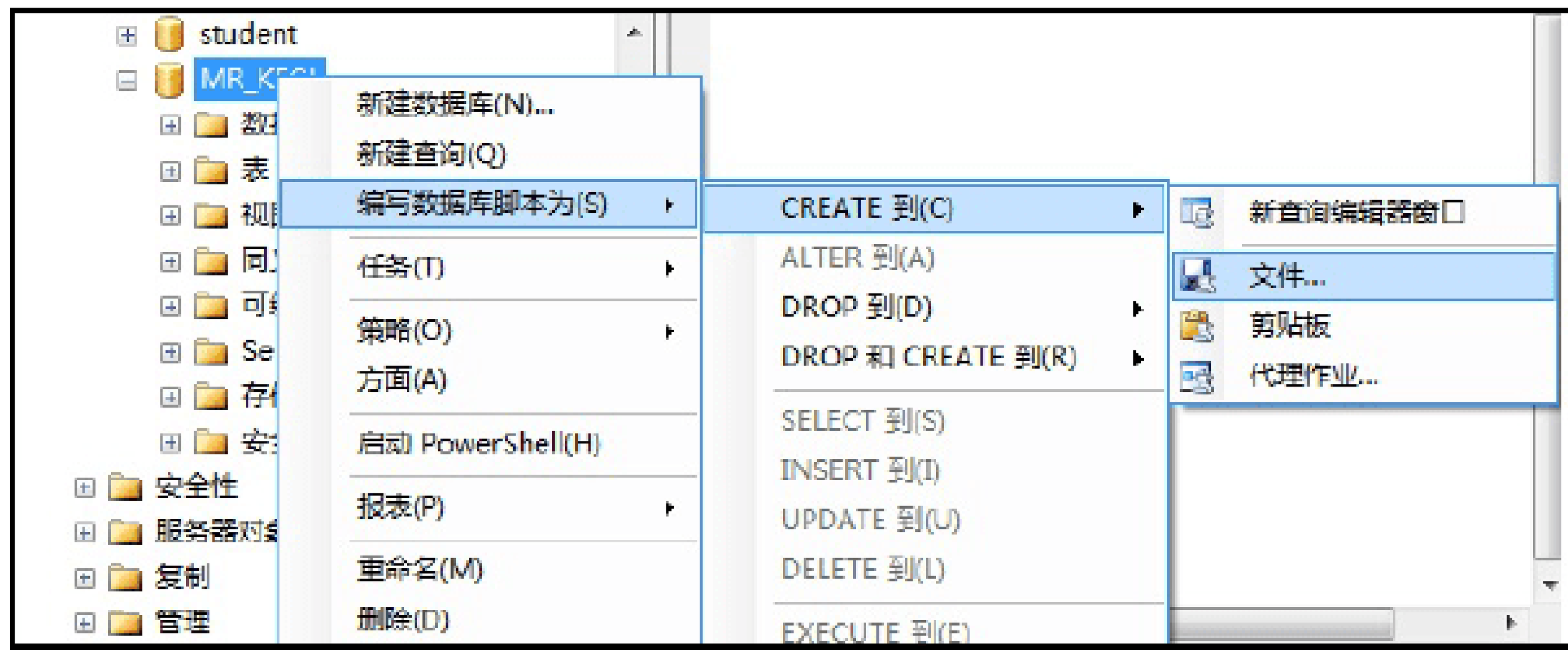


图 19.40　编写脚本模式

（3）进入“另存为”对话框，如图 19.41 所示。在“文件名”文本框中输入相应的脚本名称，单击“保存”按钮，开始编写 SQL 脚本。

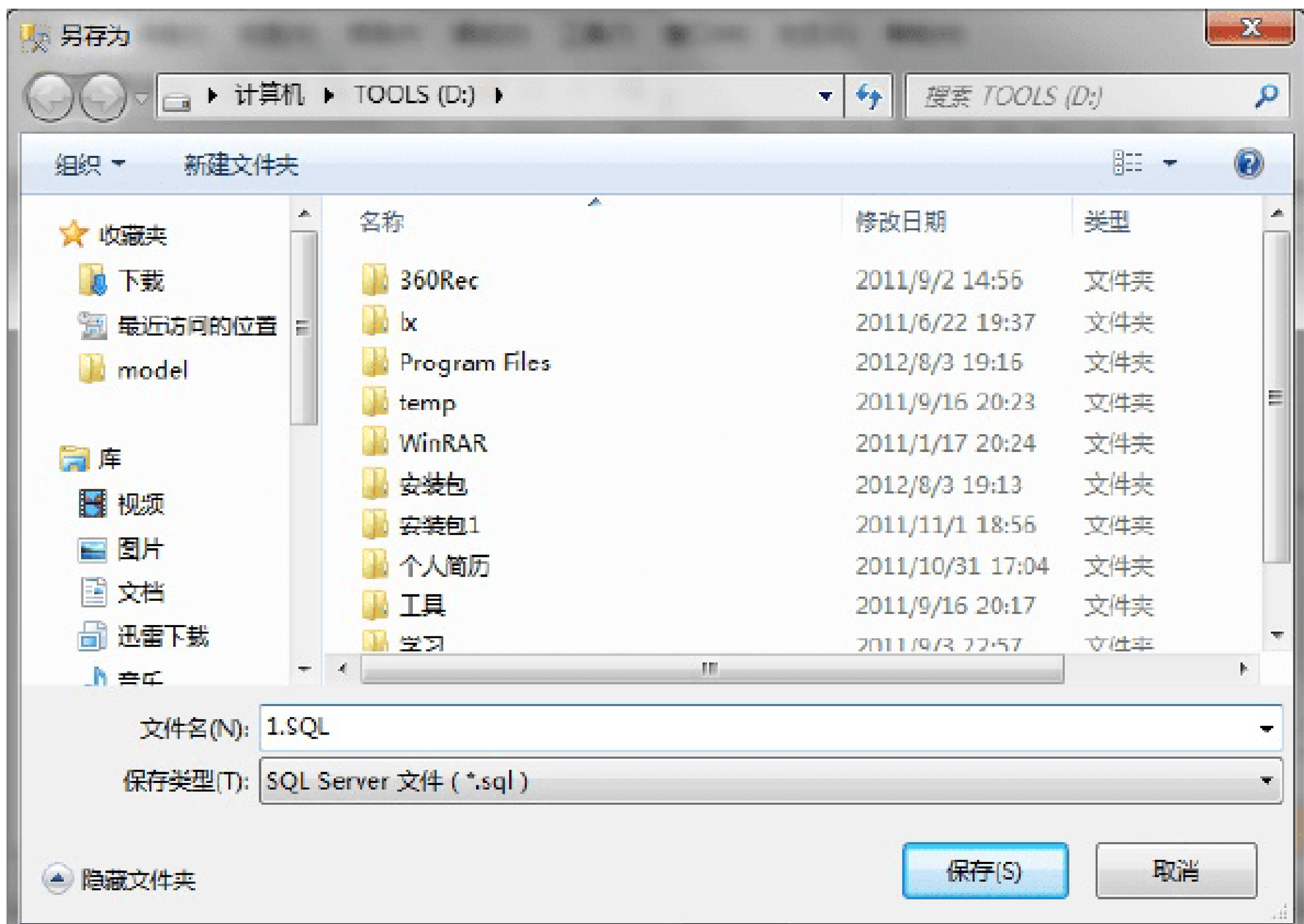


图 19.41　生成脚本

19.5.2　将数据表生成脚本

除了将数据库生成脚本文件以外，用户还可以根据需要将指定的数据表生成脚本文件。下面将数据库 student 中的数据表 course 生成脚本文件。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）展开指定的数据库 student→“表”选项。

（3）鼠标右键单击数据表 course，在弹出的快捷菜单中选择“编写表脚本为”→“CREATE 到”→“文件”命令，如图 19.42 所示。

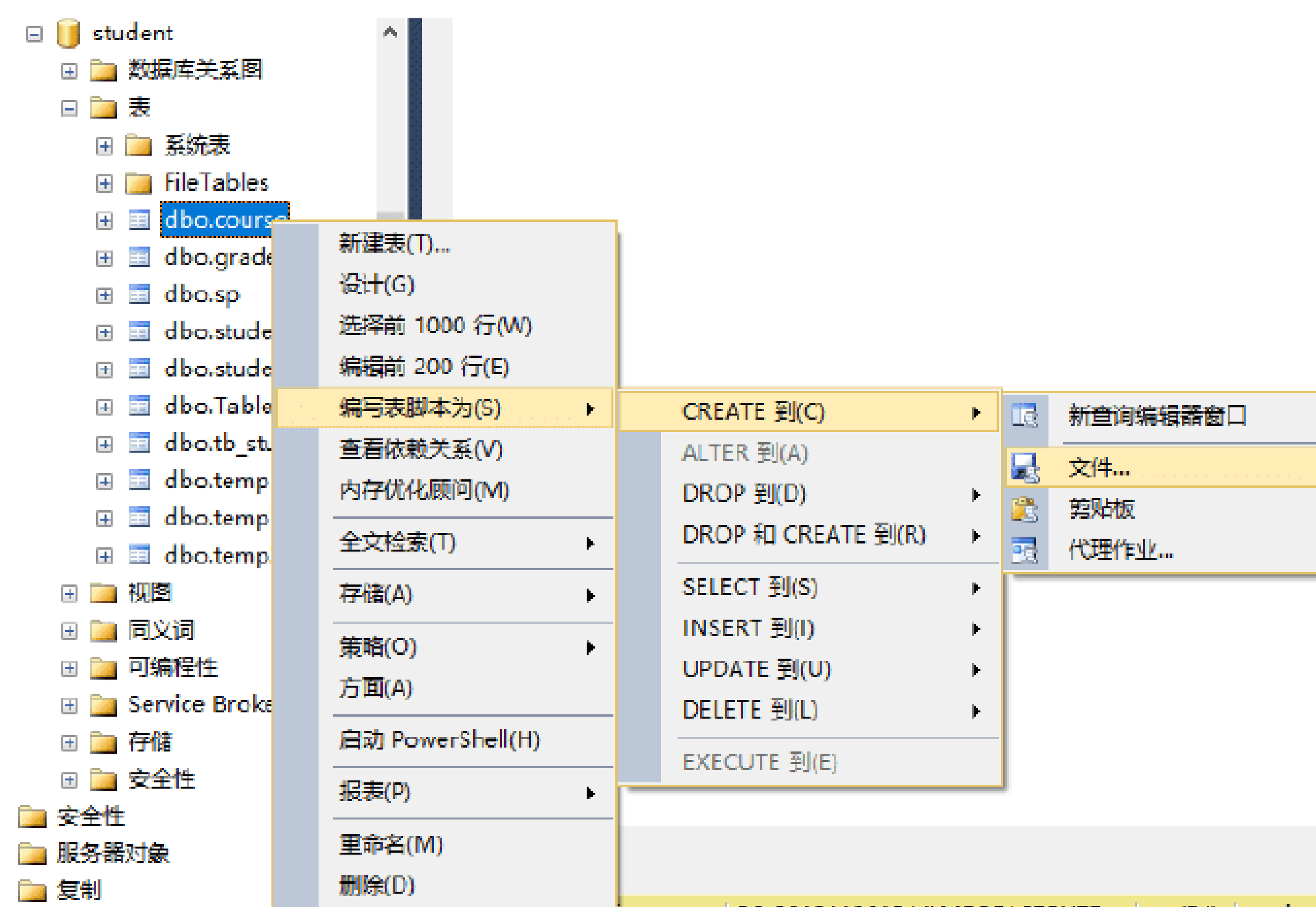


图 19.42　编写脚本模式

（4）进入“另存为”对话框，如图 19.43 所示。选择脚本保存位置，在“文件名”文本框中输入相应的脚本名称，单击“保存”按钮，开始生成 SQL 脚本。

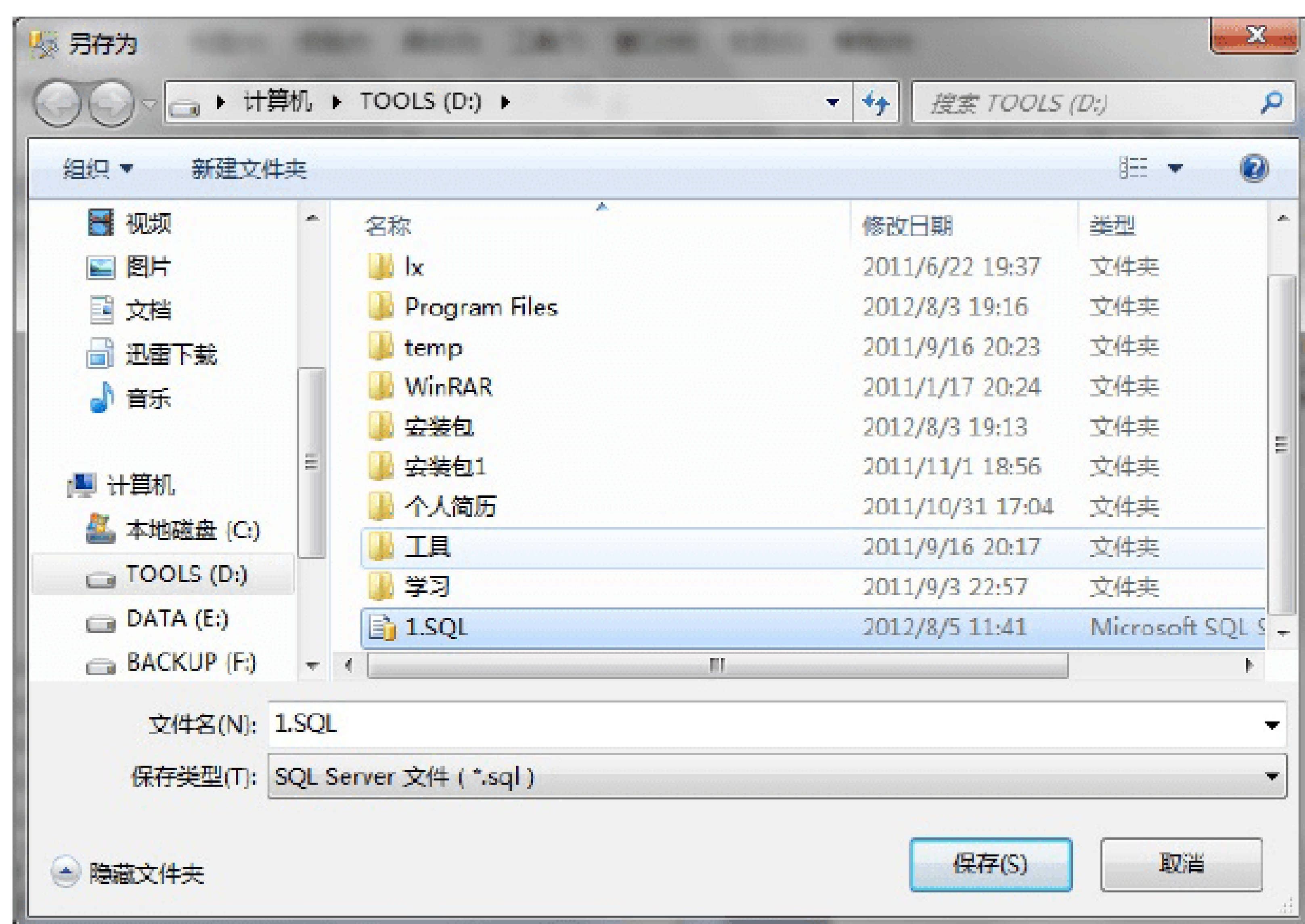


图 19.43　生成脚本

19.5.3　执行脚本

脚本文件生成以后，用户可以通过SQL Server Management Studio工具对指定的脚本文件进行修改，然后执行该脚本文件。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）选择“文件”→“打开”→“文件”命令，弹出“打开文件”对话框，从中选择保存过的脚本文件，单击“打开”按钮。脚本文件就被加载到 SQL Server Management Studio 工具中了，如图 19.44 所示。

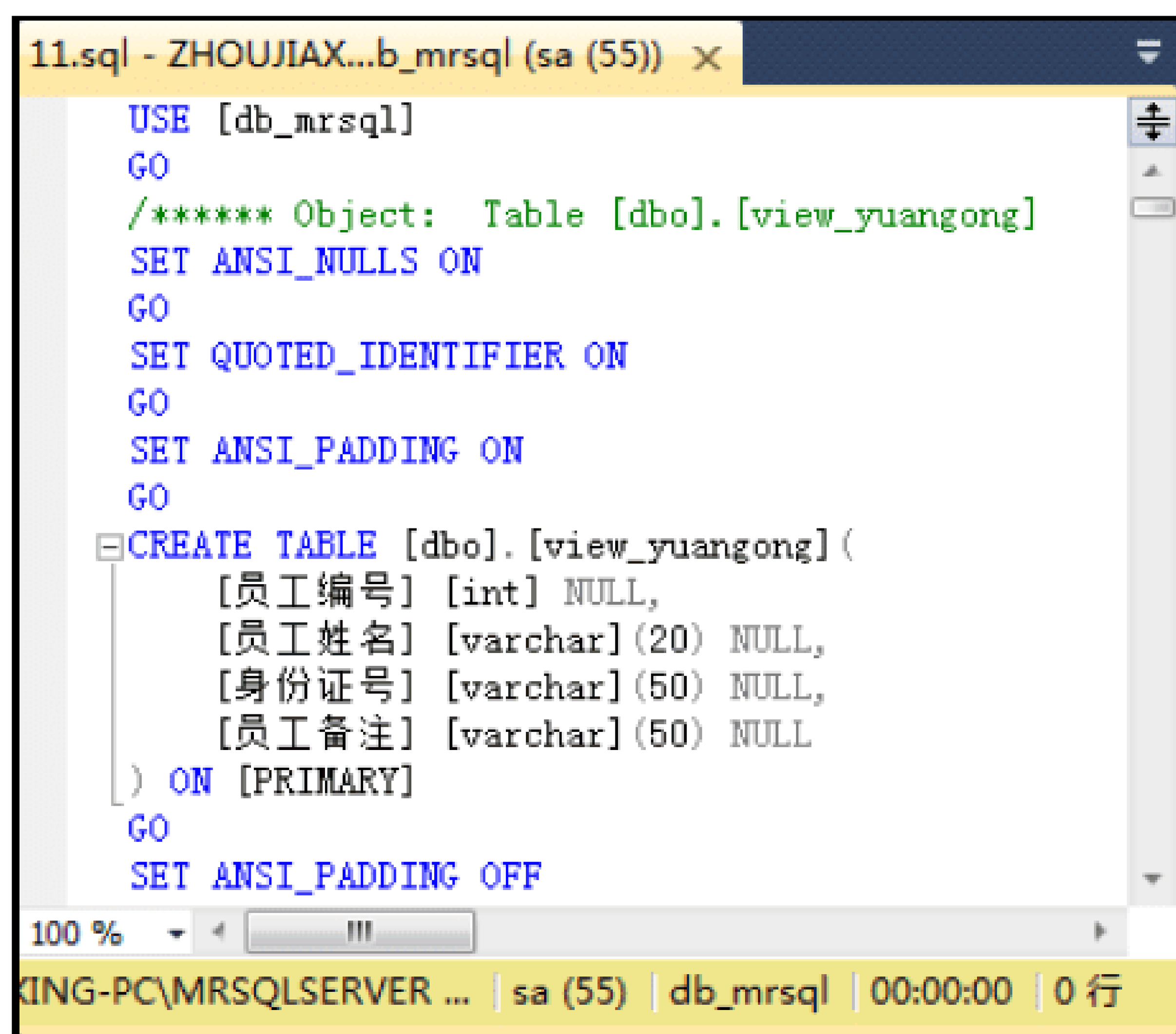


图 19.44　脚本文件

（3）在打开的脚本文件中可以对代码进行修改。修改完成后，可以按 Ctrl+F5 快捷键或✓按钮首先对脚本语言分析，然后按 F5 键或! 执行(X)按钮执行脚本。


视频讲解

19.6　数据库维护计划

数据库在使用的过程中必须进行定期维护，如更新数据库统计信息，执行数据库备份等，以确保数据库一直处于最佳的运行状态。SQL Server 2014 提供了维护计划向导，通过它读者可以根据需要创建一个维护计划，生成的数据库维护计划将对从列表中选择的数据库按计划的间隔定期运行维护任务。

下面将通过维护计划向导创建一个维护计划，名为“MR 维护计划”，完成对数据库 books、MR_KFGL 和 MR_Buyer 的维护任务（包括数据库检查完整性及更新统计信息）。具体操作步骤如下。

（1）启动 SQL Server Management Studio 工具，并连接到 SQL Server 2014 中的数据库。在“对象资源管理器”中展开“管理”节点。

（2）鼠标右键单击“维护计划”选项，在弹出的快捷菜单中选择“维护计划向导”命令，如图 19.45 所示。

（3）进入“SQL Server 维护计划向导”界面，如图 19.46 所示。

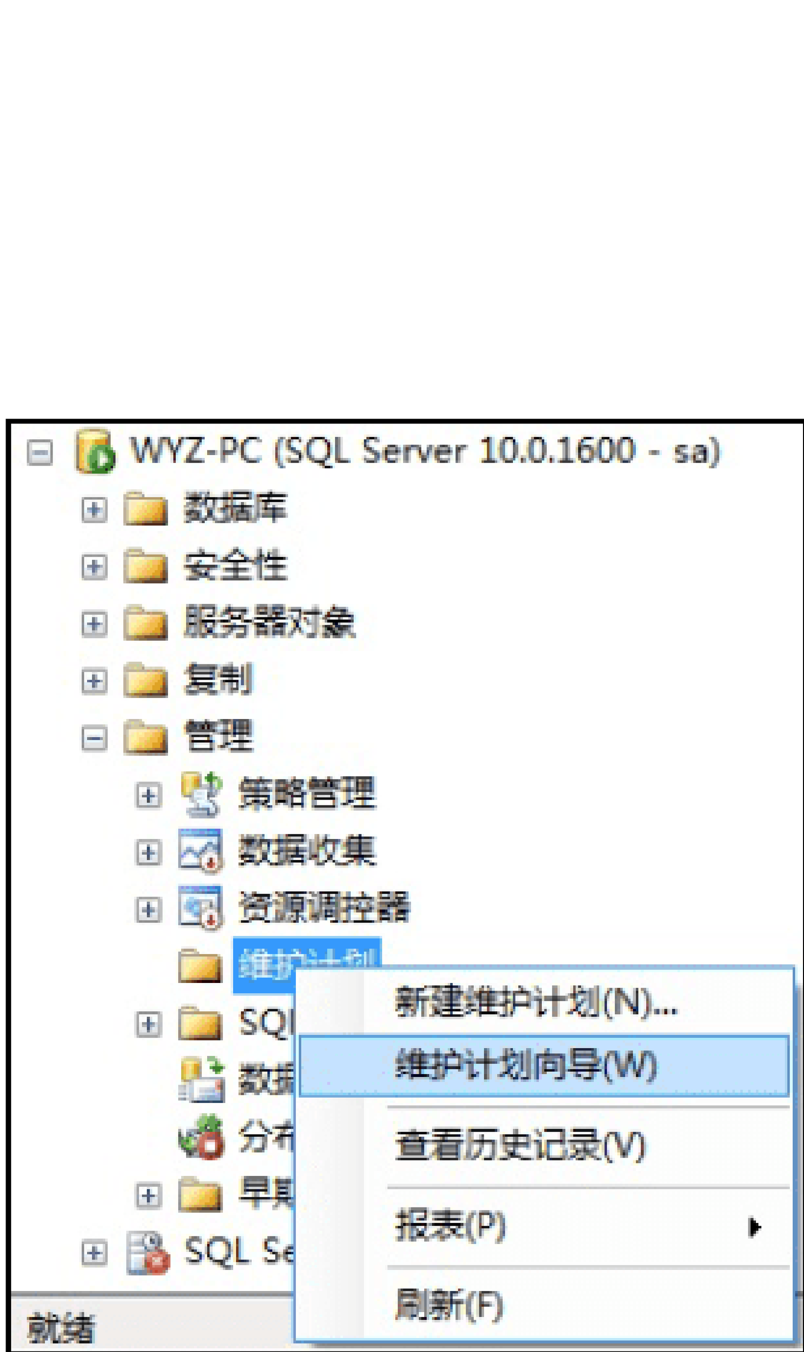


图 19.45　新建维护计划



图 19.46　SQL Server 维护计划向导

（4）直接单击“下一步”按钮进入“选择计划属性”界面，在“名称”文本框内输入维护计划的名称“MR 维护计划”，如图 19.47 所示。

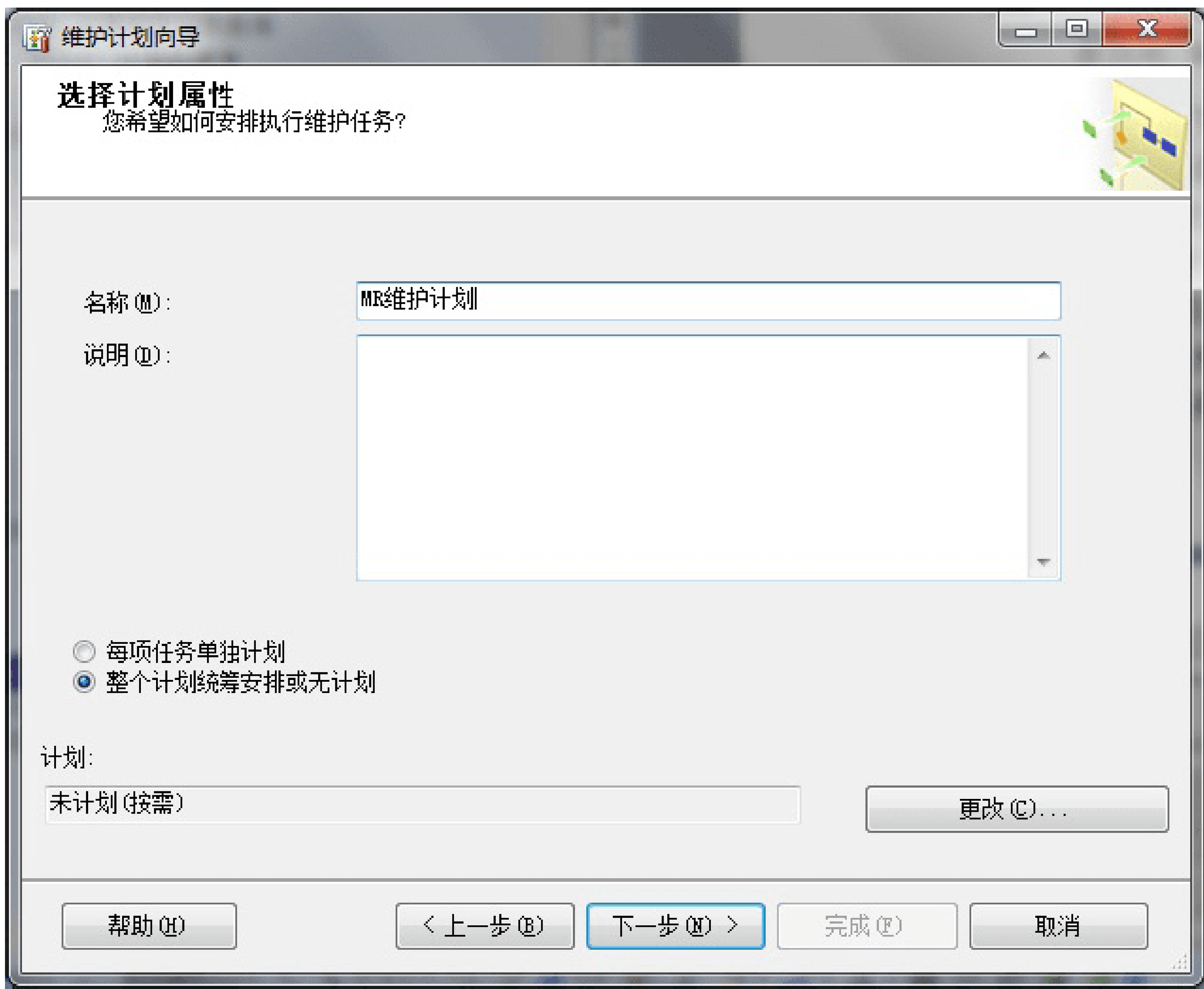


图 19.47　选择计划属性

（5）单击“下一步”按钮，进入“选择维护任务”界面，如图 19.48 所示。从列表框中选择一项或多项维护任务。这里选择“检查数据库完整性”和“更新统计信息”选项。

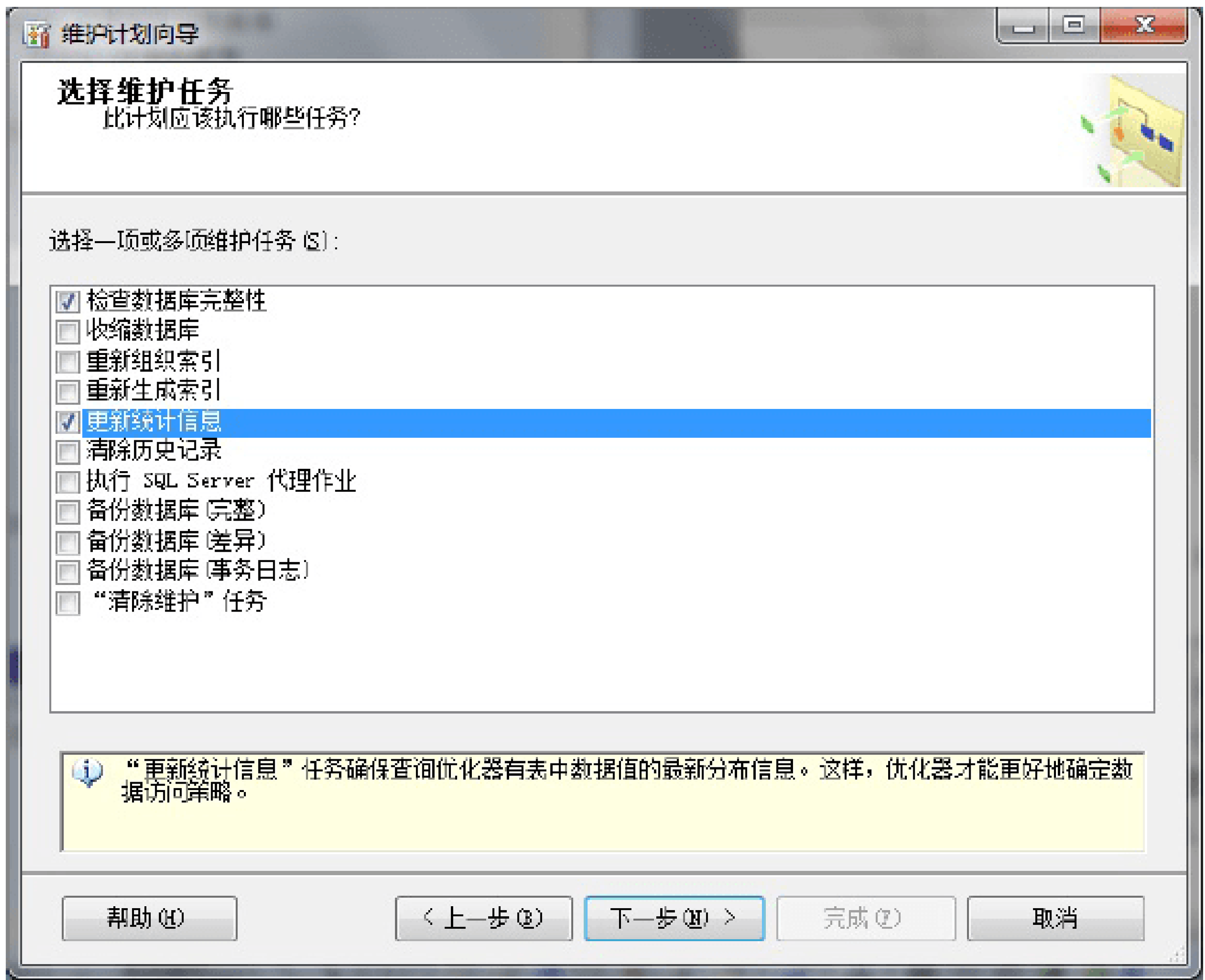


图 19.48　选择维护任务

（6）单击“下一步”按钮，进入“选择维护任务顺序”界面，如图 19.49 所示。在该界面中选择维护任务，通过单击“上移”和“下移”按钮可以调整执行任务的顺序。

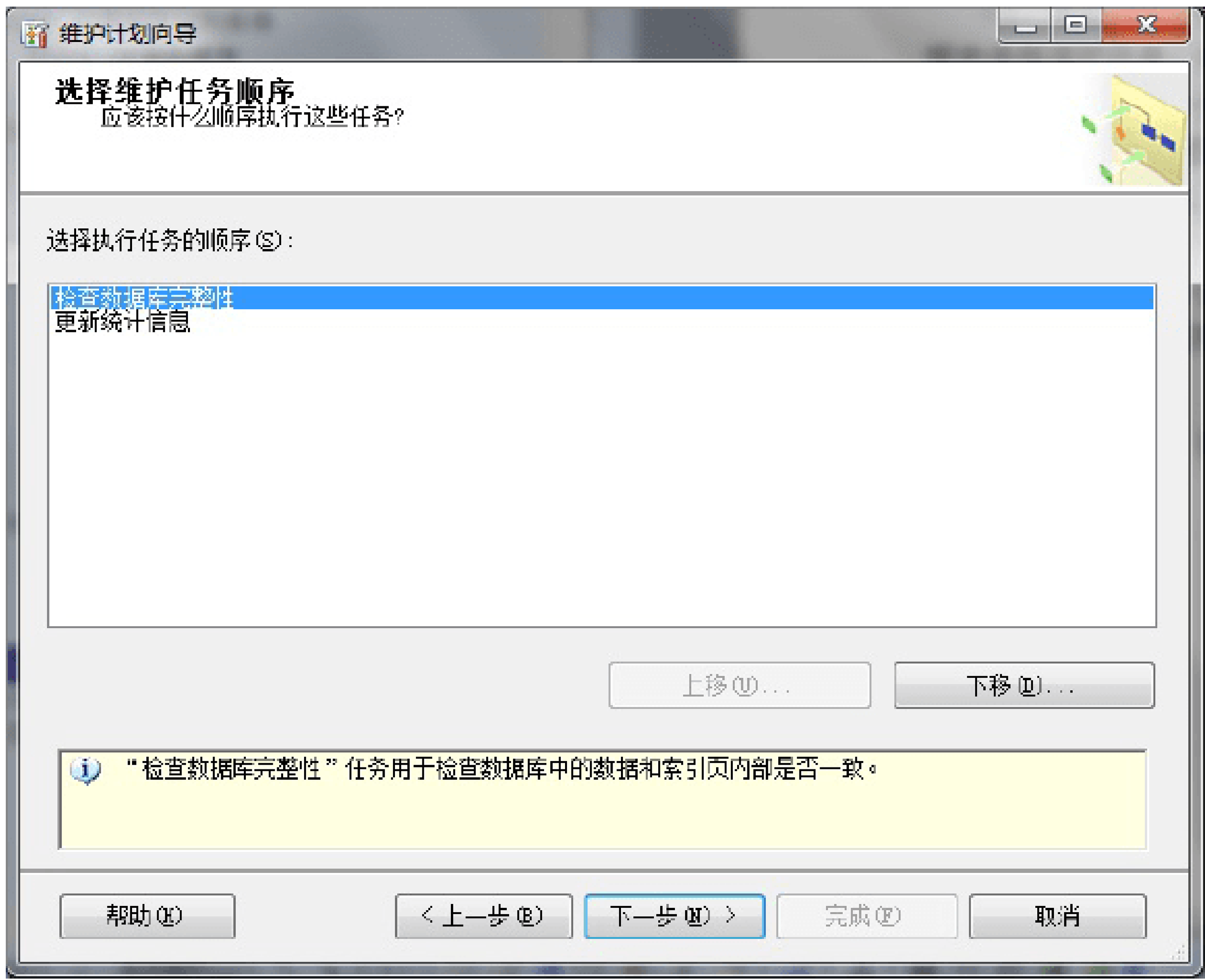


图 19.49　选择维护任务顺序

（7）单击“下一步”按钮，进入“配置维护任务”界面，这里要配置的维护任务是“数据库检查完整性”。在“数据库”下拉列表中选择任意一种数据库对其进行维护。这里选中“以下数据库”单选按钮，从中选择数据库 books、MR_Buyer 和 MR_KFGL。

（8）单击“下一步”按钮，进入“配置维护任务”界面，这里要配置的维护任务是“更新统计信息”，按照同样的操作选择特定的数据库 books、MR_Buyer 和 MR_KFGL 进行维护。

（9）单击“下一步”按钮，进入“选择计划属性”界面。

（10）单击“确定”按钮，进入“定义‘更新统计信息’任务”界面，如图 19.51 所示。

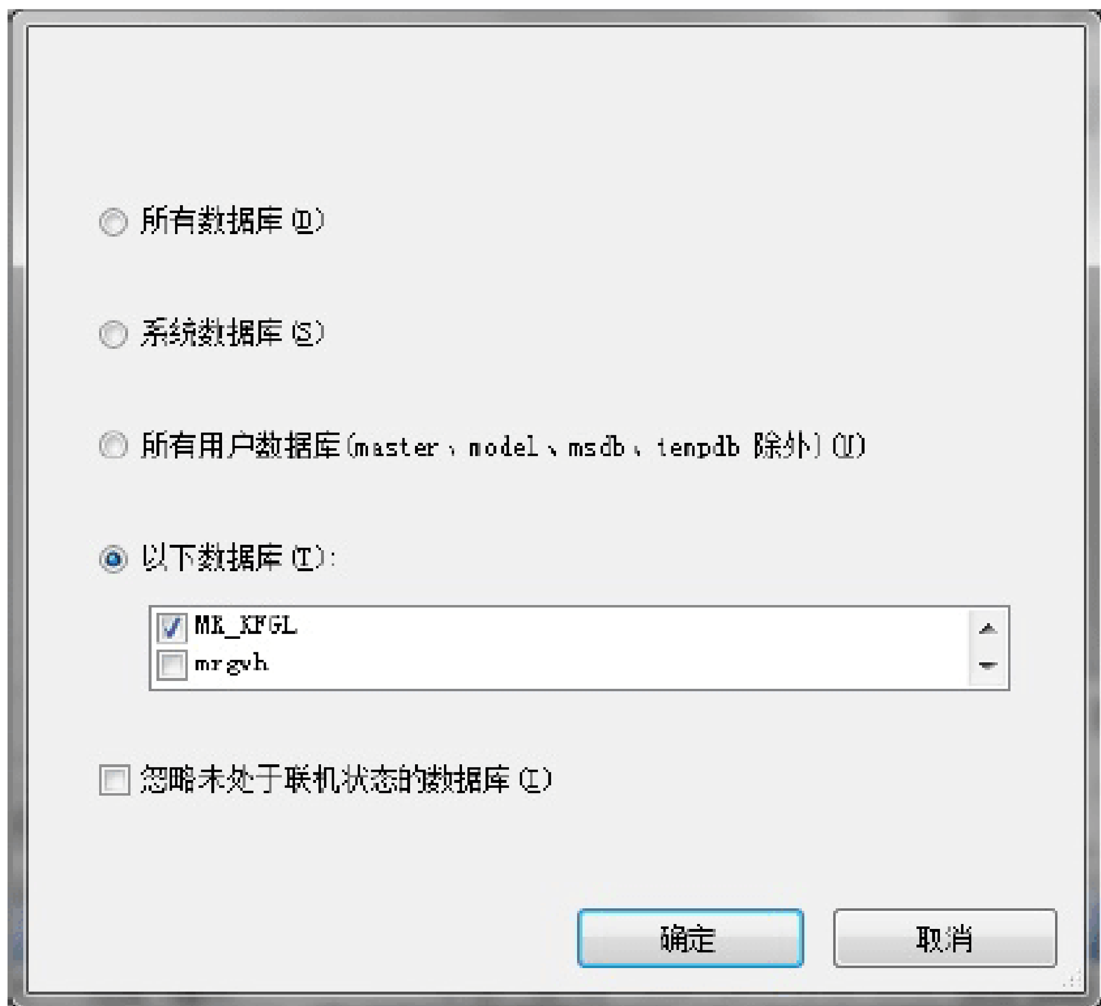


图 19.50　配置维护任务

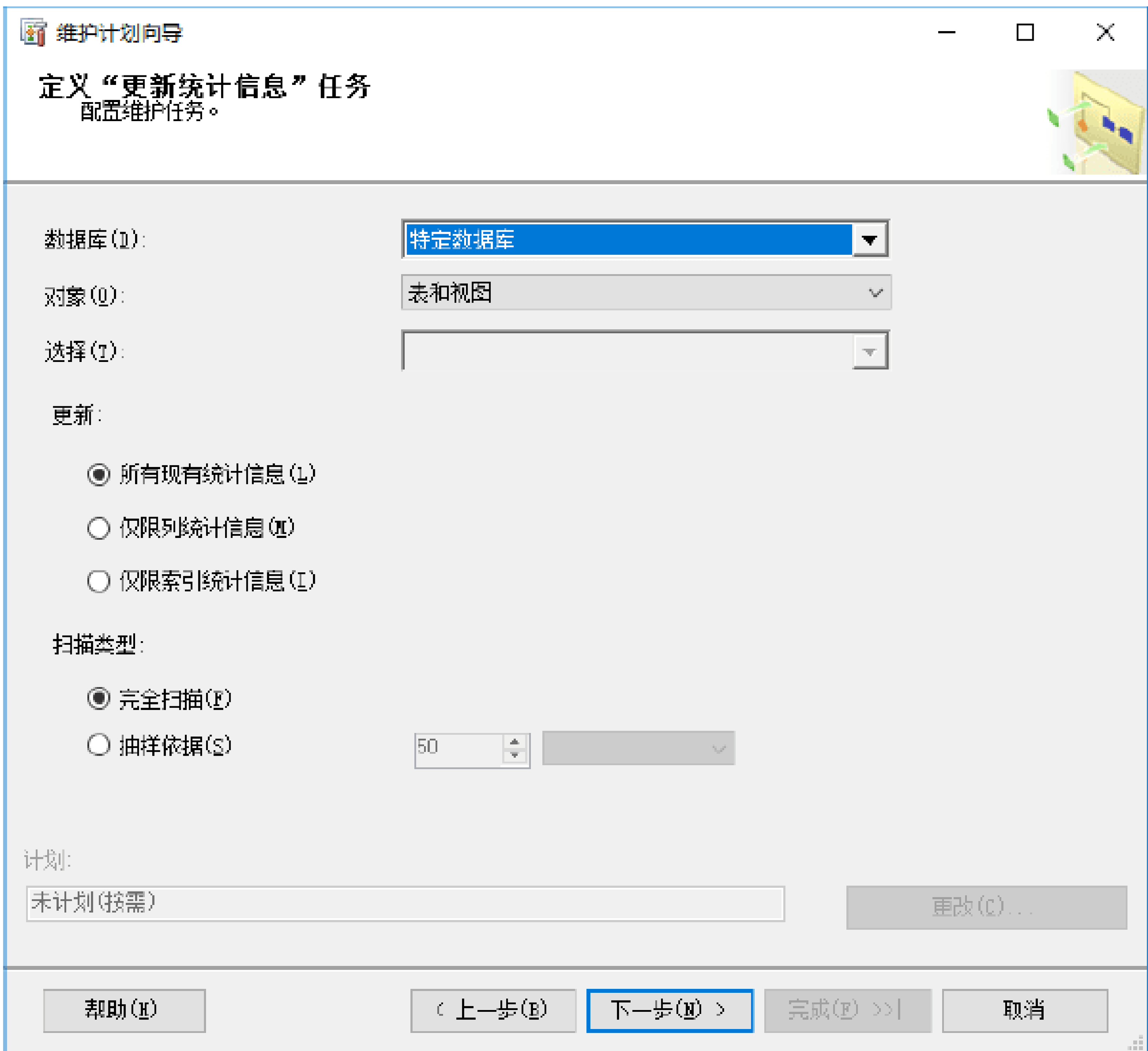


图 19.51　定义“更新统计信息”任务

（11）单击“下一步”按钮，进入“选择报告选项”界面，如图 19.52 所示。通过该界面对维护计划选择一种方式进行保存或分发。这里选中“将报告写入文本文件”复选框，单击其后的 ... 按钮，选择保存位置。

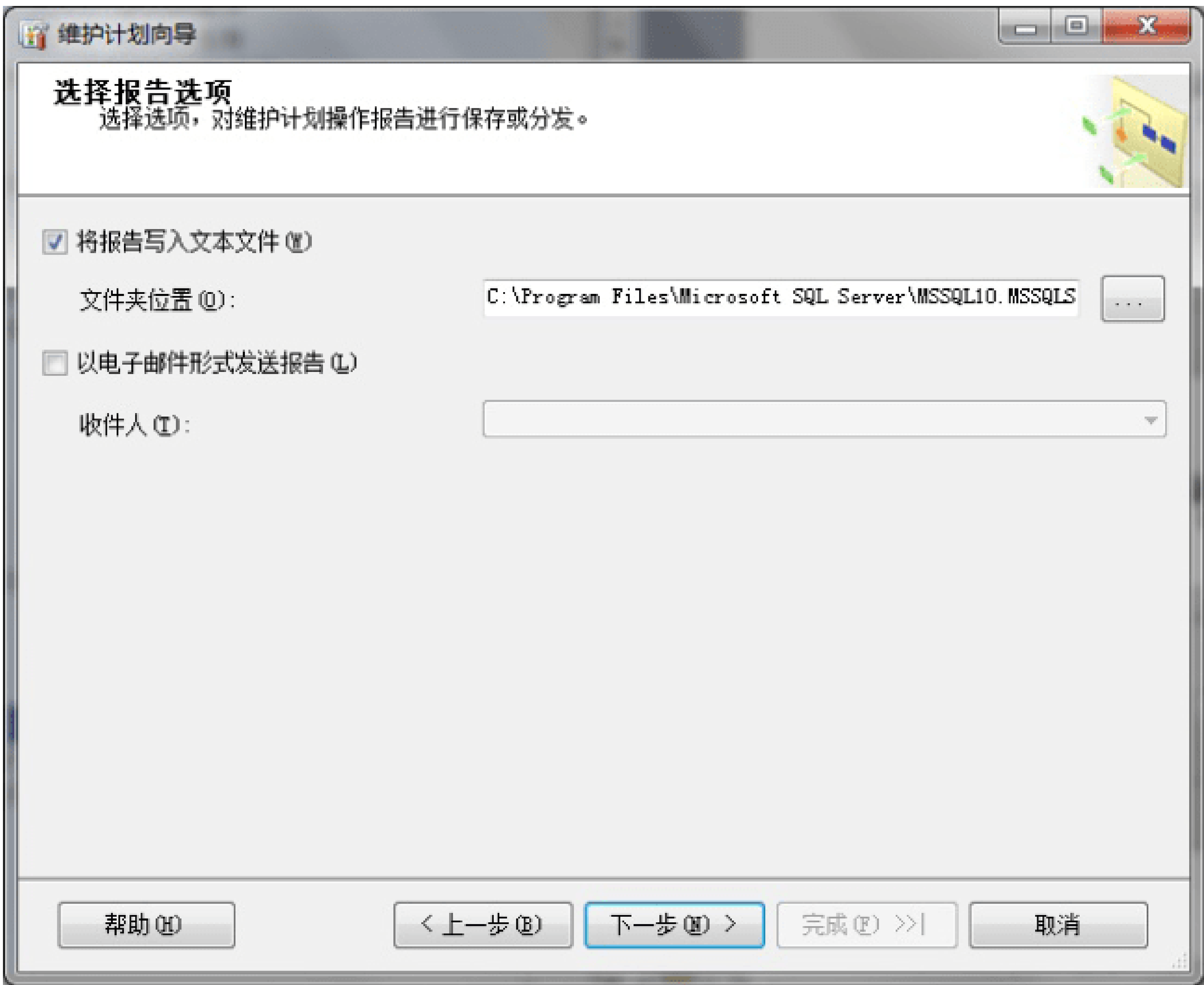


图 19.52　选择报告选项

（12）单击“下一步”按钮，进入“完成该向导”界面，如图 19.53 所示。该界面列出了维护计划中创建的相关选项。

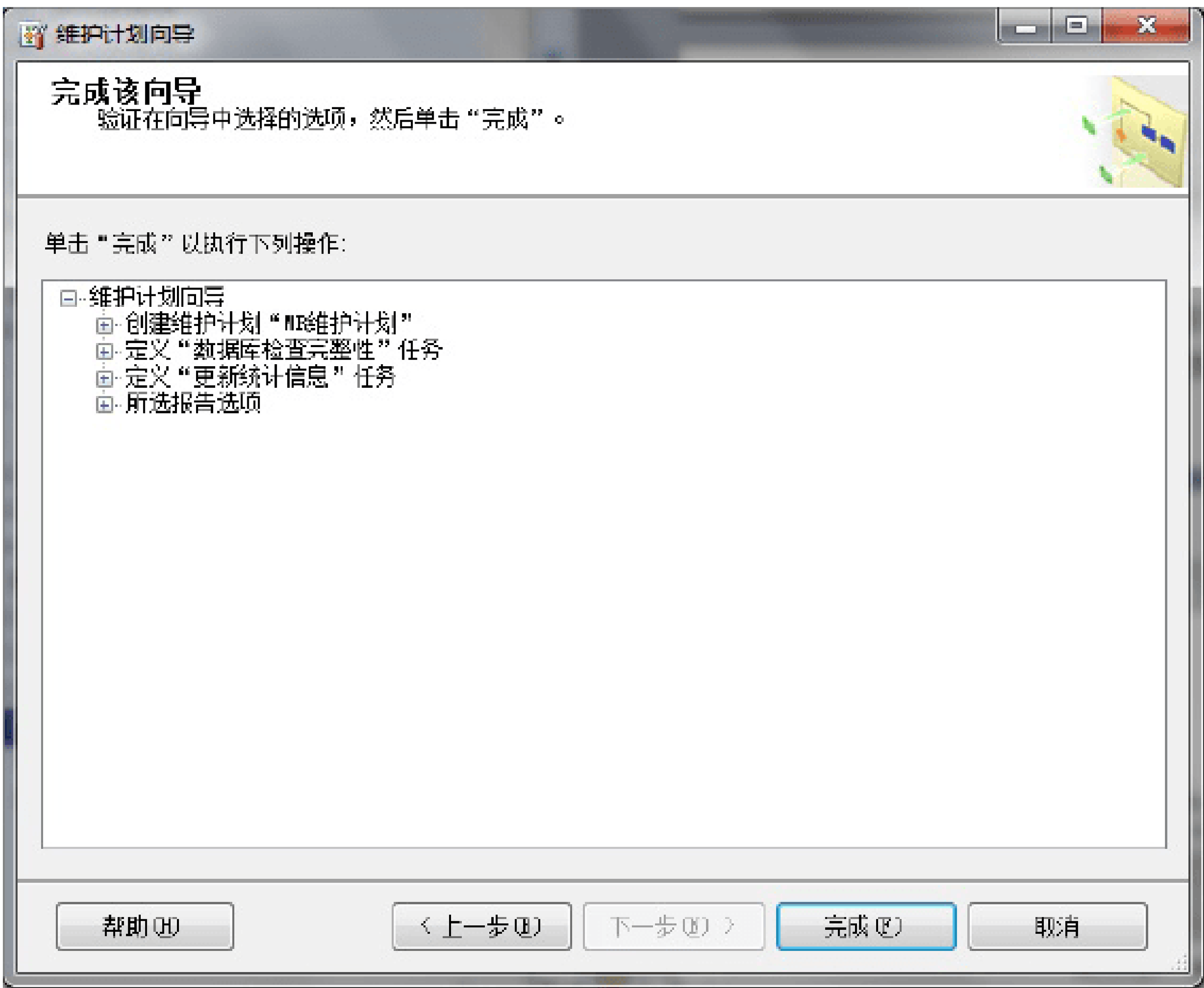


图 19.53　完成该向导

（13）单击“完成”按钮，维护计划向导开始执行，执行成功后单击“关闭”按钮即可，如图 19.54 所示。

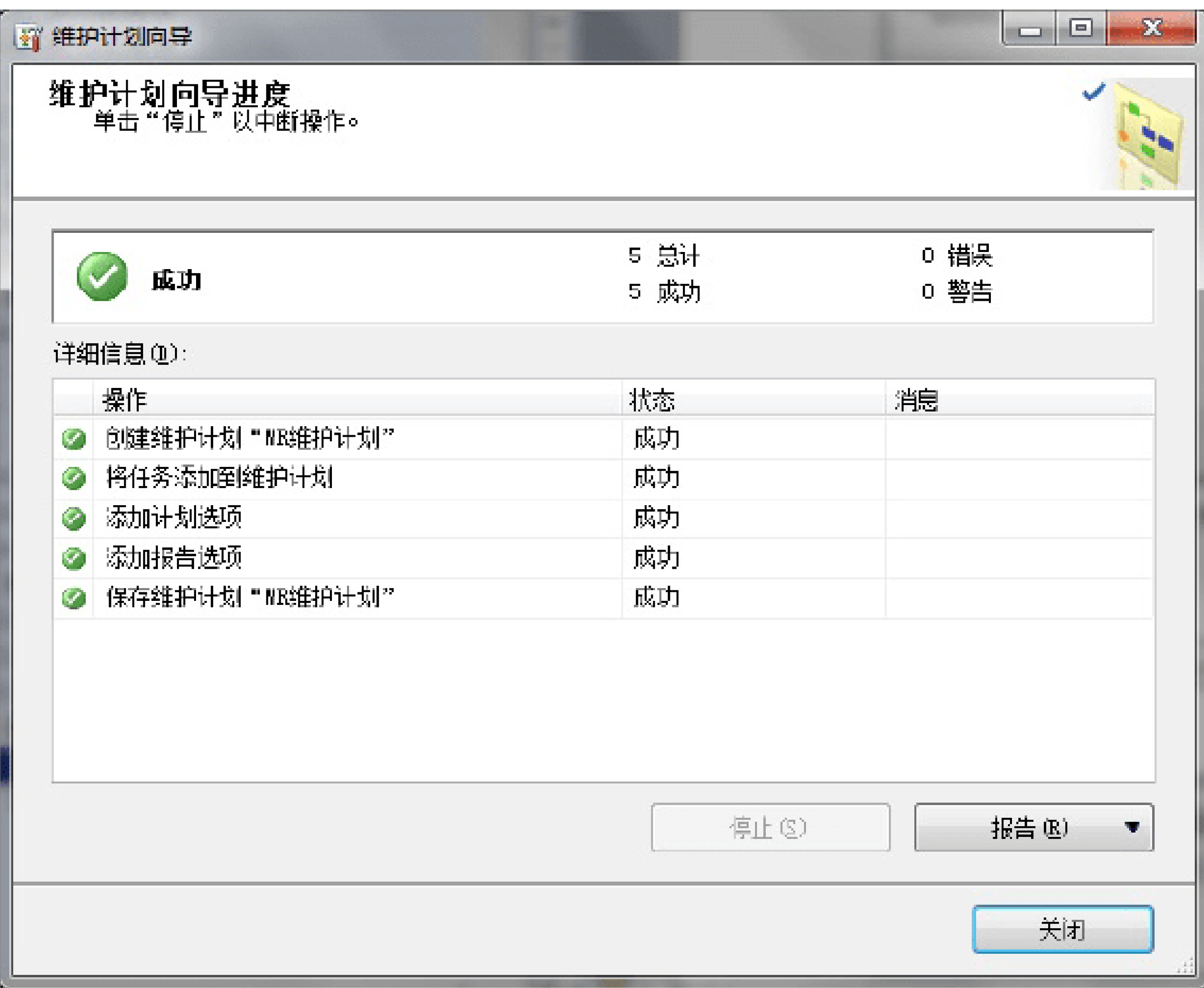


图 19.54　维护计划向导进度

19.7　小　　结

本章介绍 SQL Server 2014 中对数据库及数据表的维护管理。读者应熟练掌握脱机与联机数据库、分离和附加数据库、导入和导出数据表、备份和恢复数据库等操作，能够执行将数据库或数据表生成脚本的操作，了解数据库维护计划。